Universidad Nacional de La Plata
Facultad de Informática

Trabajo de Grado

Estudio del proceso unificado de desarrollo de software mediante una aplicación ejemplo. Comparación del mismo con una metodología tradicional.

Carrera: Licenciatura en Informática

Autor: De Caroli, Alicia Susana

Directora: Pons, Claudia

La Plata, Marzo de 2003

A mis queridos padres y esposo, Chicha, Carlos y Pablo

Indice de contenido

Capítulo 1 - Introducción.

Capítulo 2 - Análisis de requerimiento.

Primer ciclo de desarrollo.

Capítulo 3- Casos de Uso.

Capítulo 4 - Modelo Conceptual.

Capítulo 5 - Diagrama de secuencias del sistema

Capítulo 6 - Contratos.

Capítulo 7 - Diagrama de colaboración

Capítulo 8 -Revisión del Caso de Uso ImputarPagos.

Capítulo 9- Diagramas de colaboración finales y Conexión con la interfase del usuario.

Capítulo 10 - Diagrama de Clases

Segundo Ciclo de desarrollo

Capítulo 11- Descripción del Ciclo de Desarrollo

Capítulo 12- Casos de uso

Capítulo 13 - Modelo Conceptual

Capítulo 14 - Diagramas de secuencias y contratos

Capítulo 15 - Diagramas de colaboración

Capítulo 16 - Diagrama de Clases

Comparaciones y conclusiones

Capítulo 17- Comparación de Metodologías de trabajo

Anexos.

Capítulo 1 - Introducción

1.1- Necesidad de proceso de desarrollo de Software

La construcción de aplicaciones de software; tiende, en la actualidad, a ser cada vez más y más compleja y grande. Esto se debe al rápido desarrollo del mundo informático y a las características cambiantes del mundo actual.

Desde el punto de vista del desarrollo informático, cabe mencionar los siguientes puntos:

- *El continuo y gran avance tecnológico y a la vez el abaratamiento de las máquinas. Esto lleva a que una gran masa del mercado tenga acceso a equipamiento muy poderoso (hoy en día, no solo las grandes corporaciones tienen máquinas muy potentes) y que los usuarios esperen más de él.*
- *El gran avance en productos de desarrollo de aplicaciones de software. Estos productos son cada vez más sofisticados y más poderosos.*
- *El uso creciente de Internet que permite en intercambio de todo tipo de información (texto sin formato, fotos, diagramas, y multimedia).*

Estos aspectos, reducen en gran medida muchas de las limitaciones de equipamiento y de software que en otro momento se pudo haber tenido. Pero, a su vez, hacen que los usuarios, al ver nuevas posibilidades, aumentan sus exigencias; y soliciten sistemas cada vez más y más sofisticados, que se adapte mejor a sus necesidades (esto implica mayor complejidad).

Desde el punto de vista de las características cambiantes del mundo actual, lo que queremos decir es que lo que hoy es un sistema útil para una corporación, puede que mañana se convierta en obsoleto. Las aplicaciones que desarrollamos deben continuamente cambiar, y no porque lo que estamos haciendo este mal, sino porque los requerimientos de nuestros usuarios cambian, y si no las cambiamos, inevitablemente las estaremos llevando a ser inservibles. De esta manera, debemos tratar de construir sistemas fácilmente cambiables y adaptables a las necesidades cambiantes de cliente, y a la vez que sean extendibles (se le puedan agregar nuevas funcionalidades sin que para ello tengamos que modificar toda la lógica).

Estas dificultades crecientes y tendencia a la complejización hacen a la necesidad de estudiar distintos procesos de desarrollo de software; los cuales no solo involucran código entendible por una máquina sino que atraviesa por todos los ciclos de la vida del sistema, desde la concepción, pasando por el diseño, codificación, mantenimiento y evolución hasta llegar al retiro.

1.2- Proceso de desarrollo.

Un proceso efectivo proporciona normas para la realización de aplicaciones eficientes y de calidad. Además guía a los distintos participantes (desarrolladores, clientes y usuarios) de un proyecto.

El método debe ayudarnos a conseguir aplicaciones con características tales como extensibilidad, cambiabilidad y reusabilidad y a descubrir, tempranamente, los riesgos que pueden llevar al fracaso del software.

El proceso de desarrollo de software está compuesto por varias actividades diferentes. La pregunta que nos surge es: ¿cómo deben organizarse estas actividades para lograr la calidad deseada en el producto final?

Entre todas las metodologías existentes, describiremos 2:

- *Cascada*
- *Proceso Unificado*

1.2.1- Metodología de desarrollo de software de Cascada

Constituye una metodología ampliamente usada. La idea es dividir el proceso en una secuencia de etapas:

- *Estudio de costos y beneficios*
- *Análisis de requerimientos y especificación*
- *Diseño*
- *Codificación y prueba de módulos*
- *Integración y prueba del sistema*
- *Instalación*
- *Mantenimiento*

La salida de una etapa constituye la entrada de la próxima. Este proceso supone la realización completa de la etapa en una sola vez. Lo que queremos decir es que cuando se está en la etapa de diseño (por ejemplo) se la realiza para todo el proyecto y solo se pasa a la siguiente fase cuando ésta ha concluido con el diseño.

El aporte del proceso de cascada a la ingeniería de software, fue muy importante ya que ayudó a que los desarrolladores de software comprendieran dos aspectos importantes:

- *que el proceso de desarrollo de software debe estar sujeto a disciplina, organización y planeamiento*
- *que la codificación del producto debe posponerse hasta que los objetivos estén bien definidos*

Sin embargo este proceso posee algunas dificultades:

- Los requerimientos son especificados al principio del proyecto y paradojalmente, es al final del mismo cuando se tiene la claridad suficiente como para definir lo que se quiere hacer.

- Asume que una vez que los requisitos son definidos, ellos no cambiarán más.

- *Existe un énfasis en la elaboración de documentación, lo que hace difícil poder visualizar las características del sistema en grandes aplicaciones (donde hay gran cantidad de texto escrito).*

1.2.2- Metodología de desarrollo de software del proceso unificado

Una metodología más contemporánea en el desarrollo de software es la del proceso unificado.

El proceso unificado es un método para organizar las actividades relacionadas a la creación, descubrimiento y mantenimiento del sistema de Software.

Otra definición podría ser que es un proceso de desarrollo de software o sea un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. Un proceso unificado es un marco genérico de trabajo que puede especializar una gran variedad de sistemas.

La idea de este proceso es la de dividir un proyecto en funcionalidades más pequeñas (llamadas mini - proyectos) que se resolverán en su totalidad de a una o unas pocas a la vez. Dividimos al sistema en subsistemas más fáciles de resolver.

1.2.2.1- Cualidades del proceso unificado

- ***Tiene en cuenta las características cambiantes de los requerimientos:*** *en cada iteración se examinan los requerimientos, incorporando los cambios necesarios al sistema.*

- ***Estudia en los primeros ciclos de desarrollo los módulos más significativos*** *(aquellos que su no resolución puede ocasionar el fracaso del sistema). Los criterios de elección de los módulos a resolver primero son:*
  - *por su complejidad*
  - *porque posee gran variedad de conceptos*
  - *porque poseen conceptos que se utilizarán en otras funciones*
  - *porque el cliente desea que se resuelva primero*

  Al seleccionar primero aquellas partes del proyecto en donde se encuentran los mayores riesgos potenciales, logramos resolverlos en etapas tempranas, cuando aún no nos apremian los tiempos de entrega y cuando su resolución es menos compleja.

- ***Existe mayor comunicación entre los usuarios y los desarrolladores:*** *al examinar a las aplicaciones en pequeñas porciones manejables, los usuarios tienen una visión más clara de lo que se está desarrollando, puesto que pueden ver los resultados al final de cada iteración. Por lo cual es muy probable que se encuentren funcionalidades que no eran tenidas en cuenta al principio y a la vez, se modifiquen otras por no cumplir con lo que el cliente desea de ellas.*

1.2.2.2- Características del proceso unificado

Posee 3 características distintivas y definitorias:

1 – Esta guiado por casos de uso
2 – Esta centrado en la arquitectura
3 – Es iterativo e incremental

1.2.2.2.1- Esta guiado por los Casos de Uso

Para la construcción de un sistema exitoso debemos conocer lo que todos los actores necesitan y desean. Un actor es alguien o algo que interactúa con el sistema que se está desarrollando. Un caso de uso es una funcionalidad que proporciona un resultado de valor a un actor. Representan los requisitos funcionales.

La especificación funcional tradicional responde a la pregunta ¿Qué debe hacer el sistema?. La estrategia de la metodología de los casos de uso agrega 3 palabras a esta pregunta: para cada usuario. Esta 3 palabras nos fuerzan a pensar en los usuarios y no solo en las funciones que debieramos tener.

Pero los casos de uso no son solamente una herramienta para especificar requisitos, también guían el proceso de desarrollo. De esta manera cuando se lo realiza, sus desarrolladores:

- *Crean e implementan una serie de modelos que llevan a cabo los casos de uso.*
- *Revisan cada modelo para que satisfagan los casos de uso.*
- *Prueban la implementación y garantizan que se haya realizado lo que se describe en los casos de uso.*

Así los casos de uso no solo guían al descubrimiento de los requisitos, sino que proporcionan un hilo conductor a través del desarrollo de toda la aplicación. Dirigido por casos de uso quiere decir que el proceso de desarrollo sigue un hilo, avanza a través de un flujo de trabajo que comienza por los casos de uso. Es decir los casos de uso se especifican, se diseñan y finalmente son fuentes para que los ingenieros de prueba construyan los casos a comprobar.

1.2.2.2.2- Centrado en la arquitectura

La arquitectura es una vista completa del diseño (dejando de lado los detalles), muestra las características más importantes de la aplicación.

Una forma de comprender el significado de la arquitectura es realizar una analogía con el esqueleto humano cubierto de músculos, entre los huesos y la piel. Consideremos un esqueleto constituido por unos pocos músculos(software) sólo los necesarios para permitir que el esqueleto haga movimientos básicos. El sistema es el cuerpo entero con el esqueleto, piel y todos los músculos. El esqueleto constituye la arquitectura del sistema a los cuales se le agregan los músculos (software terminado) para lograr los movimientos.

La arquitectura se expresa en forma de vistas de todos los modelos del sistema, los cuales todos juntos representan al sistema en un todo. Esto implica que hay vistas arquitectónicas del modelo de caso de uso, del modelo de análisis, del modelo de diseño, del modelo de implementación y del modelo de despliegue.

Cada producto tiene tanto una función como una forma. Ninguna de las dos son suficientes por sí mismas. Estas dos fuerzas tienen que equilibrarse para obtener un producto efectivo y exitoso. En nuestro caso las funciones son especificadas por los casos de uso y la forma en que se realizan dichas funciones son especificadas por la arquitectura. Así la arquitectura y los casos de uso deben evolucionar en paralelo. Los casos de uso, por un lado, deben encajar en la arquitectura cuando ella se lleva a cabo. Por el otro lado, la arquitectura deber permitir el desarrollo de todos los casos de uso requeridos en el presente y en un futuro.

1.2.2.2.3– Es iterativo e Incremental

Como ya mencionamos anteriormente, esta metodología del proceso unificado supone la división del proceso en partes más pequeñas o miniproyectos. "Cada miniproyecto es una iteración que resulta en un incremento" [Jacobson, Booch y Rumbaugh2000]. Las iteraciones hacen referencia al flujo de trabajo, y los incrementos al crecimiento en funcionalidad del producto. En cada iteración se seleccionan un conjunto de casos de uso que aumentan, en su conjunto, las funciones del sistema. La elección se basa en inclusión de los casos de uso que poseen los riesgos más importantes. Las iteraciones sucesivas se construyen sobre los desarrollos de iteraciones anteriores, tal como quedaron en la última iteración.

Como cada iteración es un miniproyecto, se comienza con los casos de uso seleccionados, se continua con el análisis, diseño, implementación y prueba, terminando la iteración con un código ejecutable del caso de uso en desarrollo.

Así en cada iteración se realizarán las siguientes tareas:

- *Selección de los casos de uso a desarrollar*
- *Desarrollo de los mismos pasando por las etapas de análisis, diseño, implementación y prueba y teniendo en cuenta la arquitectura seleccionada como guía.*
- *Si la iteración cumple con los objetivos, se pasa a la siguiente*
- *Si la iteración no cumple con los objetivos, los desarrolladores revisan sus decisiones previas para probar con un nuevo enfoque.*

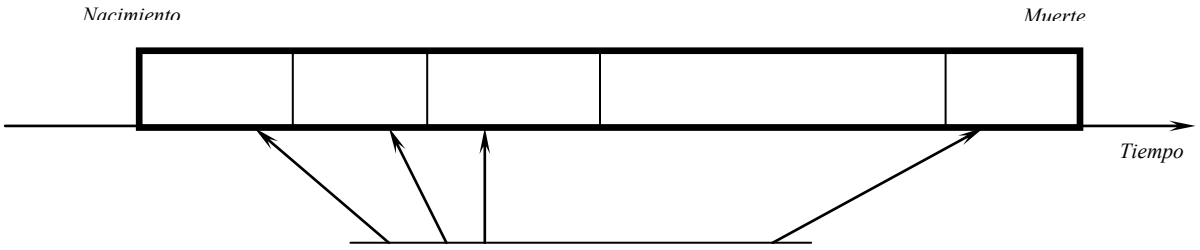
Tener en cuenta que un incremento no es necesariamente aditivo. Esto ocurre especialmente en las primera iteraciones donde los desarrolladores pueden tener que reemplazar un diseño superficial, por otro más detallado o sofisticado.

1.2.2.3- La vida del Proceso Unificado

La vida de un sistema desarrollado con el proceso unificado se desarrolla a través de una serie de ciclos, que transcurren desde su nacimiento hasta su muerte.

Cada ciclo produce una nueva versión de la aplicación, y cada versión es un producto preparado para su entrega. Por ende posee el código fuente, manuales del usuario y todos los artefactos relacionados al análisis, al diseño, a la implementación y a la prueba. Los mismos son incluidos en las entregas para posibilitar las modificaciones futuras al sistema. Tener en cuenta que los sistemas que desarrollamos, son realizados en ambientes cambiantes. Los sistemas operativos pueden mejorar, las bases de datos pueden

cambiar, las máquinas pueden ser mejoradas o cambiadas, incluso los propios requisitos de nuestro sistema pueden cambiar (a medida que los comprendemos mejor). Luego, para afrontar esta dificultad es necesaria la documentación de todos los diagramas realizados en cada ciclo.



Ciclos
Los ciclos concluyen con una versión para el cliente
La vida de un proceso consta de ciclos desde su nacimiento hasta su muerte
Figura 1.1

Cada ciclo se divide en cuatro fases:

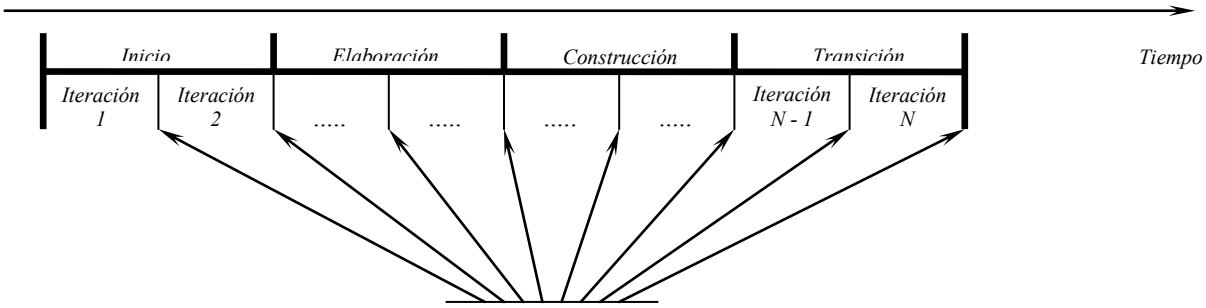
- ***Inicio****: esencialmente esta fase responde a las preguntas ¿Cuáles son las principales funciones de sistema? ¿Cómo podría ser la arquitectura del sistema? ¿Cuál es el plan de proyecto y cuánto costará desarrollar el producto?. En esta fase se identifican y priorizan los riesgos más importantes, se planifica la fase de elaboración y se estima el proyecto de manera aproximada.*

- ***Elaboración:*** *se especifican en detalle los casos de uso de mayor riesgo o críticos y se diseña la línea base de arquitectura del proyecto.*

- ***Construcción:*** *se crea el proyecto. En esta fase la línea base de la arquitectura crece hasta convertirse en un sistema completo. La descripción evoluciona hasta convertirse en un producto preparado para la entrega al usuario. Al final habremos completado la realización de todos los casos de uso (de todas las funcionalidades) que se han establecido para este ciclo de desarrollo.*

- ***Transición:*** *el producto se convierte en una versión beta, que es probada por los usuarios de mayor experiencia. Los mismos informan defectos y deficiencias del producto a los desarrolladores, quienes corrigen errores e incorporan algunas de las mejoras sugerida (en caso de que la sugerencia implique un cambio importante se relegará para el próximo ciclo de desarrollo).*

Dentro de cada fase, los trabajos son divididos en iteraciones con sus incrementos resultantes.



Versiones
Un ciclo con sus fases e iteraciones
Figura 1.2

Cada fase termina con un hito. Cada hito se determina por la finalización de un conjunto de artefactos; es decir, ciertos modelos o documentos han sido desarrollados hasta alcanzar un estado predefinido.

Los hitos permiten controlar el progreso del trabajo según se pasa por los cuatro puntos clave. Se pueden obtener datos sobre el tiempo y esfuerzo consumidos para realizar la tarea. Esta información es útil para la estimación de tiempos y recursos humanos para otros proyectos, y en el control del progreso en contraste con las planificaciones.
Recordemos que cada fase se divide en miniproyectos o iteraciones. Una iteración comúnmente pasará por 5 flujos de trabajo: requisitos, análisis, diseño, implementación y prueba.

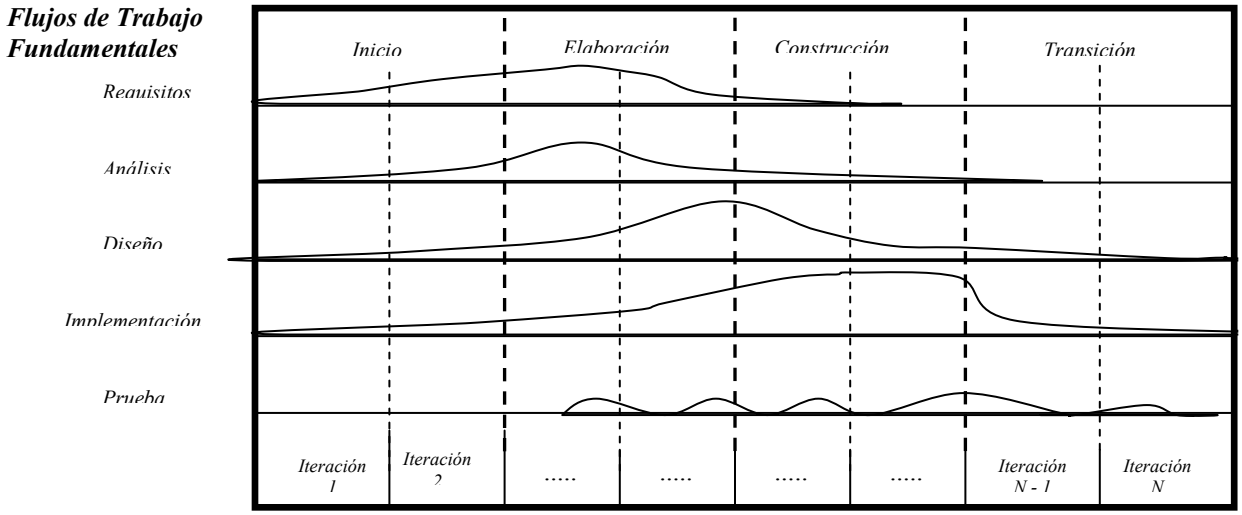


Figura 1.3

Una descripción más detallada del proceso unificado puede encontrarse en el libro "El proceso unificado de desarrollo de software" de Jacobson, Booch y Rumbaugh [Jacobson, Booch y Rumbaugh2000].

1.3- Necesidad de construcción de Modelos

De acuerdo con S. Shlaer y S.Mellor [Shlaer and Mellor88] los problemas fundamentales que entorpecen el proceso de desarrollo de software son:

- *Generalmente los desarrolladores de software no son expertos en el dominio del sistema que deben desarrollar.*

- *Los expertos en el dominio frecuentemente poseen una idea errónea, ambigua o poco clara de los requerimientos del sistema.*

- *Las personas involucradas en el proyecto (tanto usuarios como desarrolladores) poseen diferentes esquemas conceptuales del problema y utilizan diferente vocabulario para transmitir sus requerimientos.*

- *Frecuentemente la especificación del sistema se modifica durante todo su ciclo de vida.*

En los finales de los años 70 se observó un cambio importante en la filosofía del desarrollo de software, tendiente a solucionar los problemas descriptos arriba. Tom DeMarco en su libro Structured Analysis and System Specification [DeMarco 79] introdujo el concepto de ingeniería de software basada en modelos. DeMarco destacó que la construcción de un sistema de software debe ser precedida por la construcción de un modelo del sistema, tal como se realiza en otros sistemas ingenieriles. De esta forma, el modelo de un sistema provee un medio de comunicación y negociación entre usuarios, analistas y desarrolladores.

Mediante este cambio de filosofía los desarrolladores concluyeron que el punto de partida en el proceso de desarrollo de software es la construcción de un modelo, el cual actúa como una especificación precisa de los requerimientos que el sistema debe satisfacer.

Un modelo del sistema consiste en una conceptualización del dominio del problema. El modelo se focaliza sobre el mundo real: identificando, clasificando y abstrayendo los elementos que constituyen el problema y organizándolos en una estructura formal.

Durante el proceso de desarrollo de software diferentes modelos del sistema en construcción son creados. Las diferencias entre estos modelos residen en los aspectos del sistema que son contemplados (ningún modelo representa al sistema completo, sino que cada modelo hace énfasis en una parte del sistema) y en el grado de abstracción (en las primeras etapas del ciclo de vida se construyen modelos más abstractos, que luego son sustituidos y/o complementados por modelos más concretos). Por ejemplo los modelos de análisis capturan sólo los requerimientos esenciales del sistema de software, describiendo lo que el sistema hará independientemente de cómo se implemente. Por otro lado, los

modelos de diseño y los modelos de implementación describen como el sistema será construido en el contexto de un ambiente de implementación determinado (plataforma, sistema operativo, bases de datos, lenguajes de programación, interfaces, etc.).

1.3.1- Utilidades de los modelos

Los modelos de un sistema sirven para:

- *Para definir las necesidades del usuario*

- *Como un medio de comunicación y negociación entre los usuarios y los desarrolladores y entre los distintos desarrolladores entre sí.*

- *Como un documento de referencia durante la corrección de errores en el producto. Luego de introducir modificaciones en el sistema, la especificación es necesaria para chequear que la nueva implementación realmente está corrigiendo los errores contenidos en la versión previa del producto.*

- *Como un documento de referencia durante la evolución del producto. En el caso de tener que adaptar el producto debido a cambios en los requerimientos, la especificación original debe ser adaptada para reflejar estos cambios consistentemente.*

- *Como medio de detectar y resolver dicrepancias entre los distintos puntos de vista de los usuarios acerca de los requerimientos, brindando así bases firmes para las siguientes etapas de desarrollo.*

1.3.2- Enfoques para la construcción de modelos

Los dos principales enfoques para la construcción de modelos son:

- ***Algorítmico:** donde las principales piezas de un sistema son procedimientos y funciones que se ejecutan sobre estructuras de datos estáticas.*

- ***Orientado a Objetos:** donde el elemento central del sistema de software es el Objeto (o clase de objetos). Un objeto es un elemento perteneciente al dominio del problema o al dominio de la solución; una clase es una descripción de un grupo de objetos; cada objeto tiene una identidad (es decir, es distinguible de los otros objetos), tiene un estado (dado por los valores de sus atributos y relaciones) y un comportamiento (dado por la forma en que el objeto reacciona al recibir determinados mensajes). Una descripción detallada del paradigma de objetos puede encontrarse en [Budd 91, Booch 94, Shlaer and Mellor 88]*

1.3.3- Importancia de los modelos.

El modelo de un problema es esencial para describir y entender el problema, independientemente de cualquier posible sistema informático que se use para su

automatización. El modelo constituye la base fundamental de información sobre la que interactúan los expertos en el dominio del problema, los analistas y los desarrolladores de software. Por lo tanto es de fundamental importancia que exprese la esencia del problema en forma clara y precisa. Por otra parte, la actividad de construcción del modelo es una parte crítica en el proceso de desarrollo. Los modelos son el resultado de una actividad compleja y creativa y por lo tanto son propensos a contener errores, omisiones e inconsistencias. La verificación del modelo es muy importante, ya que los errores en esta etapa tienen un costoso impacto sobre las siguientes etapas del proceso de desarrollo de software.

1.3.4- Cualidades de los modelos.

Las cualidades relevantes para los modelos de análisis:

- ***El modelo debe ser claro, no ambiguo y entendible**. Para que el modelo resulte útil debe ser accesible (es decir entendible y manejable) para todos sus usuarios y debe poseer una semántica precisa.*
  Debido a la existencia de conceptos cuya interpretación semántica difiere para diferentes personas (incluyendo a las personas que crean el modelo y las personas que lo leen), si el modelo del problema es incompleto, vago o inconsistente, o si se presta a interpretaciones erróneas; entonces el resultado será un sistema de software cuya funcionalidad real difiera considerablemente de la funcionalidad esperada por sus usuarios.

- ***El modelo debe ser consistente,** es decir que no debe contener información contradictoria.*

- ***El modelo debe ser completo**, es decir que debe documentar todos los requerimientos necesarios. Debido a que los desarrolladores no poseen el conocimiento necesario para lograr una modelo completo en el principio del proceso, deben comenzar con un documento de especificación incompleta y mejorarlo a medida que adquieren experiencia. Luego es importante poder incrementar el modelo, es decir poder expandir el modelo a medida que se obtiene nueva información sobre el sistema a desarrollar.*

- ***El modelo debe ser modificable.** Se necesitan modelos flexibles, que puedan adaptarse para reflejar las modificaciones surgidas por un ambiente de desarrollo cambiante.*

- ***El modelo debe ser reusable**. El modelo de un sistema debe proveer bases para el reuso de conceptos y construcciones que se presentan en forma reiterativa en una amplia gama de problemas. El reuso permite economizar esfuerzo intelectual, tiempo y dinero.*

- ***El modelo debe ser verificable.** Existen dos aspectos a tener en cuenta:*

- *él (el modelo) en sí debe ser verificado para asegurar que cumple con las expectativas del usuario.*
- *asumiendo que es correcto, puede usarse como referencia para verificar la corrección de las implementaciones del sistema.*

1.4- Acerca de esta tesis

1.4.1- Objetivo

En esta tesis, desarrollamos el análisis y diseño de los primeros ciclos de una aplicación ejemplo "Subsistema de Liquidación y pagos del Impuesto a los Automotores" utilizando el paradigma orientado a objetos y la metodología del proceso unificado de desarrollo.

Dicha aplicación ya se encuentra en funcionamiento y su propósito es la emisión de boletas de liquidación del impuesto y el control del pago del mismo. Sus desarrolladores utilizaron el paradigma algorítmico y la metodología de desarrollo de "Cascada".

Luego utilizando la experiencia obtenida de la realización de la aplicación en ambas metodologías de desarrollo y mediante la comparación de las mismas, encontramos las ventajas del proceso unificado de desarrollo.

No pretendemos realizar la implementación del proceso, ya que nuestro objetivo es observar la forma en que se estructuran internamente las aplicaciones realizadas por una u otra metodología. Consideramos que esto puede verse realizando el análisis y diseño, sin necesidad de ahondar en la implementación, cuyos detalles pueden llevarnos a desvirtuar el objetivo que pretendemos para esta tesis.

1.4.2- Organización

La parte que resta de esta tesis se encuentra organizada de la siguiente manera:

En el capítulo 2, realizamos una breve descripción de la aplicación a desarrollar y el análisis de requerimiento de todo el sistema (metas, objetivos, funciones, etc.).

En el capítulo 3 obtuvimos los casos de uso de la aplicación, los escribimos en formato de alto nivel, dibujamos el diagrama de caso de uso de toda la aplicación, seleccionamos los casos de uso que desarrollaremos en el primer ciclo de desarrollo y los escribimos en formato extendido.

Los capítulos 4 al 10 fueron dedicados para desarrollar el análisis y diseño del primer ciclo de desarrollo. Luego realizamos el diagrama del modelo conceptual, el diagrama de secuencias de los casos de uso, los contratos del sistema, el diagrama de colaboración y el diagrama de clases.

En los capítulos 11 a 16 realizamos el análisis y diseño del segundo ciclo de desarrollo.

Por último en el capítulo 17 encontramos las ventajas del proceso unificado a través de la comparación del análisis y diseño obtenido en los capítulos previos, con el análisis y diseño de la aplicación ya existente.

Capítulo 2 – Análisis de requerimiento

2.1- Acerca de la organización del capítulo

Para realizar un análisis de requerimiento que sea comprensible por el lector creemos conveniente explicar, primero, en que consiste "El Impuesto a los Automotores".

Luego realizaremos el análisis de requerimientos de toda la aplicación, describiendo las metas, objetivos, clientes y funciones que el usuario pretende para el sistema.

Tener en cuenta que en este nivel no estudiaremos un proceso particular del desarrollo de software (no estudiaremos una funcionalidad en particular de la aplicación), sino que examinaremos al sistema de software en forma global. Recién sobre el final del capítulo 3 seleccionaremos los casos de uso más significativos para comenzar con el desarrollo de los mismos (teniendo en cuenta las asunciones en el primer ciclo de desarrollo).

2.2- Descripción del Sistema

El Impuesto a los Automotor es una obligación impositiva de todo individuo propietario de un vehículo.

Si bien es anual el pago del mismo se desdobla en tres cuotas (también llamados anticipos), cada una de las cuales tiene un vencimiento. Una vez pasada la fecha límite de pago, el monto de la cuota sufre un interés que se va incrementando día a día.

El sistema del Impuesto a los Automotores se encarga de administrar los datos de vehículos y de contribuyentes (propietarios de vehículos). La suma de esta información hace factible la cobrabilidad del impuesto.

- *Los datos del vehículo tales como:*
  - *Fecha de vigencia*
  - *Marca*
  - *Modelo año*
  - *Tipo*
  - *Peso*
  - *Carga*

  determinan la forma de generar los cargos a cobrar, ya sea por la valuación de la marca o por el tipo, peso y carga.

- *Los datos del contribuyente tales como:*
  - *Apellido y nombre o razón social*
  - *Domicilio*
  - *CUIT*
  - *Tipo y numero de documento*

  permiten identificar al contribuyente y hacer posible el envío de las boletas de emisión general y/o de intimaciones por falta de pago.

- *Los datos de los pagos tales como:*
  Importe pagado
  Fecha de pago
  Forma de pago
  hacen a la correcta intimación por falta de pago y a la recaudación del impuesto.

Nota: *Puna descripción más detallada sobre el sistema se encuentra en el Manual de Usuario del Impuesto a los Automotores [Manual].*

2.3- Análisis de Requerimiento

Los requerimientos constituyen una descripción de las necesidades o funcionalidades deseadas para el producto a desarrollar.

La correcta identificación de requerimientos es esencial para comprender cual es el alcance de nuestro sistema y las necesidades de los usuarios. Una identificación incorrecta o ambigua hace al desarrollo de sistemas con funcionalidades no deseadas o que realizan parcialmente su objetivo.

La meta principal de la fase de requerimientos es la de identificar y documentar cuales son las necesidades reales del usuario, en forma que sean comprendidas claramente por los usuarios y los desarrolladores del sistema.

En las secciones subsiguientes realizaremos:

1- *La descripción del objetivo.*
2- *La identificación de los clientes del sistema.*
3- *La descripción de la meta a la que se pretende llegar con el sistema.*
4- *La identificación y categorización de las funciones del sistema.*

2.3.1- Descripción del objetivo del sistema

El objetivo es la creación de un sistema de pago e imputación del Impuesto a los Automotores usado para registrar el cumplimiento de la obligación Impositiva.

2.3.2- Identificación de los clientes del sistema.

- *Organizaciones dedicadas a realizar la recaudación del Impuesto.*
- *Entes Recaudadores*

2.3.3- Metas del sistema

- *Rápida atención al contribuyente (individuo que debe pagar el impuesto).*
- *Rápido y exacto análisis de la deuda.*
- *Rápida y automática asignación de los pagos realizados.*

- *Rápido asentamiento de los datos del vehículo y contribuyente.*

- *Rápida realización de movimientos sobre el auto o contribuyente (transferencia, exención, cambio de radicación, cambio de domicilio postal o fiscal).*
- *Rápida obtención de datos de un auto (modelo, titular, domicilio postal o fiscal).*
- *Rápida obtención de datos sobre pagos de años – cuotas de un vehículo.*

2.3.4- Funciones del sistema

Referencia	*Categoría*	*Descripción*
1.1	*Evidente*	*Registrar las cuotas adeudadas a abonar (en formato año - cuota)*
1.2	*Evidente*	*Calcular el total de la liquidación corriente de un determinado contribuyente incluyendo impuestos, interés por mora, diversos cálculos necesarios en la liquidación.*
1.3	*Evidente*	*Mostrar todas las cuotas (en formato año – cuota) con el monto de la deuda para permitir la selección de las mismas.*
1.4	*Evidente*	*Calcular el valor de las cuotas adeudas (de acuerdo a las características del vehículo).*
1.5	*Oculto*	*Registrar los importes pagados y la forma de pago una vez que el mismo ha sido completado.*
1.6	*Oculto*	*Log de las liquidaciones realizadas.*
1.7	*Evidente*	*El operador del sistema debe logonearse y poner su password para usar el sistema.*
1.8	*Oculto*	*Proveer mecanismos de almacenaje persistente.*
1.9	*Oculto*	*Proveer mecanismos de conexión con otros subsistemas.*
1.10	*Evidente*	*Desplegar información sobre:* *1- El vehículo (modelo, fecha de alta, código de marca, valuación, inciso, categoría, peso, etc.).* *2- El titular (nombre y apellido, dirección, cuit, número de documento).* *3- Cuotas a liquidar (año de la cuota a liquidar, número de cuota, fecha de vencimiento, importe original de la cuota, importe actualizado, índice de actualización de la cuota).*
1.11	*Evidente*	*Asentar los datos de un nuevo vehículo y de su titular.*
1.12	*Evidente*	*Asentar los datos del nuevo titular de un dominio, cuando se ha producido una venta.*
1.13	*Evidente*	*Asentar la exención de pago de un auto, cuando hay razones que lo jusitifiquen.*
1.14	*Evidente*	*Asentar el cese de una exención de pago.*
1.15	*Evidente*	*Modificar datos del vehículo.*
1.16	*Evidente*	*Modificar el o los domicilios fiscales.*
1.17	*Evidente*	*Modificar el domicilio postal.*
1.18	*Evidente*	*Mostrar todos los datos de un vehículo.*
1.19	*Evidente*	*Mostrar todos los datos del o los titulares y del destinatario postal.*

1.20	*Evidente*	*Mostrar los datos de las cuotas (cuotas emitidas, cuotas pagadas, monto emitido y pagado, forma de emisión y de imputación, fecha de emisión y de imputación, etc.).*
1.21	*Evidente*	*Emitir las boletas antes del vencimiento del impuesto.*
1.22	*Evidente*	*Emitir boletas de intimación según pautas del ente recaudador.*
Funciones de pago		
2.1	*Evidente*	*Manejar del pago con tarjeta de cuenta en banco capturando la información del mismo (cuenta del banco, el número de documento) vía lectora de tarjeta y autorizando la imputación por el servicio de autorización de pago del banco vía conexión de modem.*
2.2	*Evidente*	*Manejar el pago con tarjeta de crédito capturando la información de la tarjeta vía lectora de tarjeta y autorizando la imputación por el servicio de imputación de pagos de la tarjeta vía conexión de modem.*
2.3	*Oculto*	*Log del pago por tarjeta de cuenta de banco guardando la información del monto adeudado por el banco al ente recaudador.*
2.4	*Oculto*	*Log del pago por tarjeta de crédito en banco guardando la información del monto adeudado por la tarjeta al ente recaudador.*
2.5	*Evidente*	*Manejar el pago que ingresa con la información enviada por el banco.*
2.6	*Oculta*	*Log del pago por banco guardando la información del monto adeudado por el banco.*

Nota:

Hemos categorizado las funciones en:

- Evidentes: aquellas que deben ejecutarse y el usuario tiene conocimiento de las mismas.

- Ocultas: aquellas que deben ejecutarse, pero no son visibles para el usuario. Es particularmente común cuando queremos auditar una operación o almacenaje persistente de los datos.

- Opcionales: aquellas que sumarlas no afectan al costo o a otras funcionalidades.

Capítulo 3 - Casos de Uso

3.1- Acerca de la organización del capítulo

Un caso de uso es un documento narrativo que describe la secuencia de eventos de un actor (agente externo) usando un sistema para completar un proceso [Jacobson92].

En este capítulo se pretende descubrir todos los casos de uso asociados a la aplicación "Impuesto a los Automotores", para luego elegir aquellos que consideramos en el primer ciclo de desarrollo del proceso unificado de desarrollo de software. En capítulos subsiguientes desarrollaremos el análisis y diseño de los casos de uso seleccionado de acuerdo a las asunciones realizadas sobre los mismos.

Para lograr el objetivo de la sección anterior se examinarán cada uno de los puntos expuestos a continuación:

1. *Definición de los límites del sistema*
2. *Identificación de los actores*
3. *Identificación de los casos de uso*
4. *Escritura de todos los casos de uso en formato de alto nivel*
5. *Categorización los casos de uso como primarios, secundarios u opcionales*
6. *Diagrama de caso de uso*
7. *Elección de casos de uso a desarrollar en el primer ciclo de desarrollo.*
8. *Escritura de los casos de uso de mayor riesgo o influencia o críticos, en forma esencial expandida. Esto se hace para un mejor entendimiento del problema y para estimar la naturaleza y longitud del sistema.*
9. *Enunciado de las asunciones del primer ciclo de desarrollo del proceso unificado (se desarrollarán los casos de uso del punto 8).*

En el anexo "Caso de Uso" se encuentran mayores definiciones sobre los puntos expuestos en este capítulo o consultar al libro "El lenguaje unificado de modelado" de Booch, Rumbaugh y Jacobson [Booch, Rumbaugh, Jacobson99].

3.2- Definición de los límites del sistema

3.2.1 - ¿Cuales son los posibles límites de sistema a elegir?

Existen dos sistemas posibles:

1. *La terminal de software y hardware donde se realiza la liquidación y pago del impuesto.*
2. *La oficina de Rentas donde se realiza la liquidación y pago del Impuesto Automotor.*

3.2.2 - Elección del límite del sistema

En la primer propuesta el contribuyente y el operador del sistema pertenecen al ambiente externo del sistema y por ende ambos son actores del sistema (ambos usan la terminal para obtener la liquidación del impuesto).

En la segunda propuesta el operador esta dentro del ambiente del sistema y por lo tanto solo es actor el contribuyente, que arriba a la oficina de rentas para pagar si obligación impositiva.

Elegiríamos la segunda propuesta si además de estar interesados en el desarrollo de una aplicación de software, nos preocupara la reingenieria del proceso comercial (procesos u organizaciones para incrementar competibilidad o calidad).

Pero como este no es nuestro objetivo, elegimos como sistema la primer propuesta (implica solamente el desarrollo de una aplicación de software)

3.3- Identificación de los actores

A continuación, desplegaremos la lista de los actores que vemos en el sistema que estamos analizando e introduciremos una breve descripción de la tarea de cada uno de ellos.

1) ***Operador:*** *persona responsable de operar el sistema, según los datos proporcionados por el contribuyente del Impuesto*
2) ***Contribuyente:*** *Persona responsable de pagar el Impuesto al Automotor*
3) ***Sistema de Verificación de fondos de tarjeta de créditos:*** *nuestro sistema enviará a este actor datos sobre las transacciones de pago del impuesto a realizar vía tarjeta de crédito. Este organismo dará la autorización para acreditar el pago.*
4) ***Sistema de Verificación de fondos de tarjeta Bancaria:*** *nuestro sistema enviará a este actor datos sobre las transacciones de pago del impuesto a realizar vía tarjeta bancaria. Este organismo dará la autorización para acreditar el pago.*
5) ***Entidad bancaria:*** *entidad que envía datos a nuestro sistema para que él impute pagos de acuerdo a la información recibida.*
6) ***Registro del Automotor:*** *entidad que envía los datos para la realización de asentamiento o modificación de información del vehículo o de su titular (allí se presenta el contribuyente para dar de alta, exentar, transferir a un dominio)*
7) ***Ente Recaudador****: entidad responsable de dar las pautas sobre la forma de cálculo del impuesto, la forma de emisión de boletas, las condiciones que deben cumplir los vehículos que se intimarán, las estadísticas a realizar, etc.*
8) ***Implementador****: persona responsable de ejecutar los pedidos de emisión, intimación e imputación según datos proporcionados por el ente recaudador o la entidad bancaria (ejecuta los programas batch)*
9) ***Administrador del sistema:*** *persona responsable del encendido de la máquina al principio del día, de la carga de los programas pertinentes para el correcto funcionamiento del sistema, del apagado del equipo al final del día y del control de los usuarios (asignación de nuevos usuarios, revocación de usuarios, acceso de los programas a los distintos usuarios)*

3.4- Identificación de los casos de uso

3.4.1 - Método para identificar un caso de uso

Dos alternativas posibles para la elección de los casos de uso son:

a) Basados en actores:

- Identificar los actores relacionados al sistema u organización

- Para cada actor identificar los proceso que ellos inicializan

b) Basados en eventos:

- Identificar los eventos externos que el sistema debe responder

- Relacionar los eventos a actores y casos de uso

3.4.2 - Obtención de los posibles casos de uso

Basándonos en la primera aproximación, surge esta lista de casos de uso

Actores	*Funciones*
Administrador del sistema	*Start up* *Shut down*
Operador	*Login*
Operador on-line / Contribuyente / Sistema de verificación de fondos de la tarjeta de crédito / Sistema de verificación de fondos de la tarjeta de banco	*Pagar Impuesto*
Operador on-line / Contribuyente	*Liquidar Impuesto* *Consultar datos* *Cambiar el domicilio postal*
Entidad Bancaria / Implementador	*Imputar de pago*
Operador/Registro automotor	*Asentar nuevo vehículo* *Cambiar el titular* *Exentar un vehículo* *Asentar el robo de un vehículo* *Cambiar el domicilio fiscal* *Cesar una exención* *Cambio de radicación* *Cambio de datos de vehículo* *Asentar vehículo proveniente de otro lugar*
Entidad recaudadora / implementador	*Emitir boletas* *Intimar pagos* *Realizar estadísticas*

3.5- Tipo de caso de uso a utilizar

*Utilizaremos **casos de uso esenciales**, debido a que nos encontramos en la fase inicial del desarrollo de la aplicación. Estos casos de uso son más generales y nos permiten abstraernos de la tecnología que se utilizará para implementar el subsistema. Esto constituye una gran ventaja ya que no nos abrumaremos en esta etapa (en la que aún no tenemos conocimiento cabal del sistema) con las dificultades de los detalles tecnológicos, los cuales pueden tranquilamente ser resueltos en otra etapa de desarrollo (cuando tengamos bien en claro que funcionalidad debería realizar nuestro sistema). Otra ventaja de utilizar estos casos de uso es que nos permiten desarrollar aplicaciones más generales, que sirven para diversas plataformas y que por lo tanto son más reusables.*

*Por otra parte **utilizaremos formato extendido** para describir un caso de uso, cuando consideremos que este es significativo (aquellos que implican mayor riesgo su no resolución, o aquellos que poseen mayor grado de dificultad o que el usuario requiere su pronta implementación).*

***Utilizaremos el formato de alto nivel** para la descripción de los demás casos de uso. Aquellos que, por lo menos en este nivel, no consideramos tan significativos.*

La idea del proceso unificado es abordar primero el estudio de los casos de uso más significativos, para resolver lo más tempranamente posible los mayores riesgos a los que puede someterse un sistema. Estos riesgos, muchas veces, pueden conducir al fracaso o al abandono del sistema; razón por la cual es preferible encontrarlos, abordarlos y solucionarlos en etapas tempranas de desarrollo. En esta fase la inversión que se realiza para el desarrollo del nuevo software aún no es muy fuerte y además no tenemos la carga de los tiempos de entrega que nos apremie.

Nota:

En el anexo de "Caso de uso" se encuentra una breve explicación sobre las distintas formas de especificar un caso de uso. También se puede examinar el libro "Applying UML and patterns" de Larman [Larman97] para ahondar el tema.

3.6- Descripción de todos los casos de uso en un formato de alto nivel

Casos de uso	*Actores*	*Tipos*	*Descripción*
Start up	*Administrador*	*Secundaria*	*El administrador, al comienzo del día, prende la máquina y realiza las operaciones de inicialización necesarias para que se pueda comenzar a operar el sistema.*
Shut down	*Administrador*	*Secundaria*	*El administrador, al final del día laboral, realiza las operaciones pertinentes para luego poder apagar el equipamiento.*

Login	*Operador*	*Secundaria*	*El operador del sistema ingresa al mismo introduciendo su login y password. En caso de que los datos ingresados estén correctos, podrá comenzar a operar el sistema.*
Pagar Impuesto	*Contribuyente / Operador*	*Primaria*	*El contribuyente arriba a la oficina de operación para realizar el pago del impuesto. El operador registra los años cuotas que desea pagar y luego se asienta el pago. El contribuyente se retira con el comprobante de pago en mano.*
Liquidar Impuesto	*Contribuyente / Operador*	*Primaria*	*El contribuyente arriba a la oficina de operación para solicitar una liquidación del impuesto. El operador liquida los años – cuotas solicitados. El contribuyente se retira con la liquidación en mano para pagarla en el banco.*
Imputar pagos	*Entidad Bancaria / Operador*	*Primaria*	*La entidad bancaria informa sobre los pagos que se realizaron y se procede al asentamiento de los mismos. La entidad bancaria es informada de la deuda que posee con el ente recaudador.*
Asentar vehículo	*Registro Automotor / operador*	*Primaria*	*El registro automotor envía una declaración jurada con los datos de un nuevo vehículo y de su o sus titulares. El operador ingresa los datos al sistema e informa al registro.*
Cambiar la titularidad	*Registro Automotor / operador*	*Secundaria*	*El registro automotor envía una declaración jurada con los datos de la transferencia de un vehículo. El operador ingresa él o los nuevos titulares al sistema e informa al registro.*
Exentar de pago	*Registro Automotor / operador*	*Primaria*	*El registro automotor envía una declaración jurada con los datos del vehículo a ser exentado y la causa de exención (puede que de acuerdo a esto cambie la declaración jurada). El operador marca al vehículo como exento de pago a partir de la fecha de vigencia de la declaración jurada.*
Asentar el robo de un vehículo	*Registro Automotor / operador*	*Primaria*	*El registro automotor envía una declaración jurada con los datos del vehículo que fue robado. El operador marca al vehículo como exento de pago a partir de la fecha de vigencia de la declaración jurada.*
Cambiar domicilio fiscal	*Registro Automotor / operador*	*Secundaria*	*El registro automotor manda una declaración jurada con los datos del nuevo domicilio de los titulares. El operador cambia el o los domicilios fiscales e informa al registro.*
Cambiar domicilio postal	*Contribuyente / operador*	*Secundaria*	*El contribuyente se presenta a la oficina de operación y solicita que se le cambia el domicilio de emisión de sus boletas impositivas (el domicilio donde le llega la boleta). El operador asienta los datos que el contribuyente le proporciona.*
Cesar la exención	*Registro Automotor /*	*Primaria*	*El registro automotor manda una declaración jurada con los datos del motivo de cese de exención. El*

	operador		*operador marca al vehículo para que se le liquide el impuesto a partir de la fecha de cese*
Cambiar la radicación	*Registro Automotor / operador*	*Primaria*	*El registro automotor manda una declaración jurada con los datos del vehículo que realiza un cambio de radicación. El operador marca al vehículo como que a partir de la fecha no tributará más en la provincia.*
Cambiar datos del vehículo	*Registro Automotor / operador*	*Primaria*	*El registro automotor manda una declaración jurada con los cambios a realizar en los datos del vehículo. El operador asienta los cambios.*
Consultar datos	*Contribuyente / Operador*	*Primaria*	*El contribuyente se presenta a la oficina de operación para solicitar datos de un vehículo. El operador ingresa la patente y le suministra los datos del mismo. El contribuyente se retira con la información que necesitaba.*
Emitir boletas	*Ente recaudador*	*Primaria*	*El ente recaudador envía las pautas necesarias para proceder el cálculo y emisión de las boletas de la próxima cuota a vencer. Se procede a la emisión de acuerdo a la información enviada.*
Intimar al pago	*Ente recaudador*	*Primaria*	*El ente recaudador envía las condiciones que deben cumplir un vehículo para ser intimado y las pautas para emitir las boletas de intimación. Se procede a la emisión de la boleta de intimación de acuerdo a las pautas enviadas.*

3.7- Diagrama de caso de uso para el sistema de Impuesto a los Automotores

La figura 3.1 corresponde a la sección principal del diagrama de uso. En ella no especificamos las relaciones de generalización de los casos de uso, solo pusimos los casos de uso más generales y dejamos para los diagramas de caso de uso de las secciones secundarias (figuras 3.2, 3.3, 3.4 y 3.5) la especificación de la relación de generalización y de los casos de uso más específicos. La división se realizó para mejorar la lectura del diagrama.

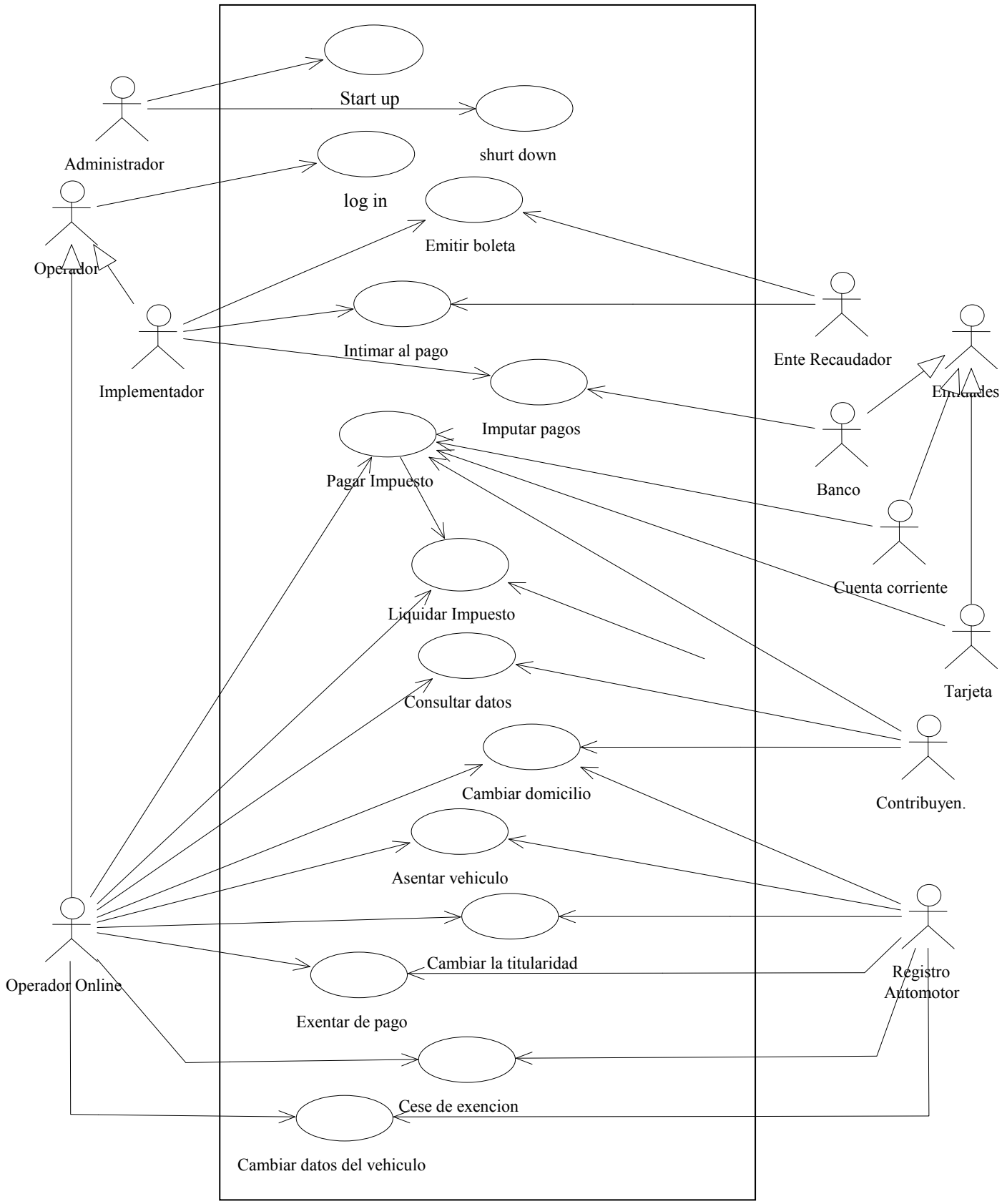


Diagrama de caso de uso de la aplicación Impuesto a los Automotores
Sección Principal
Figura 3.1

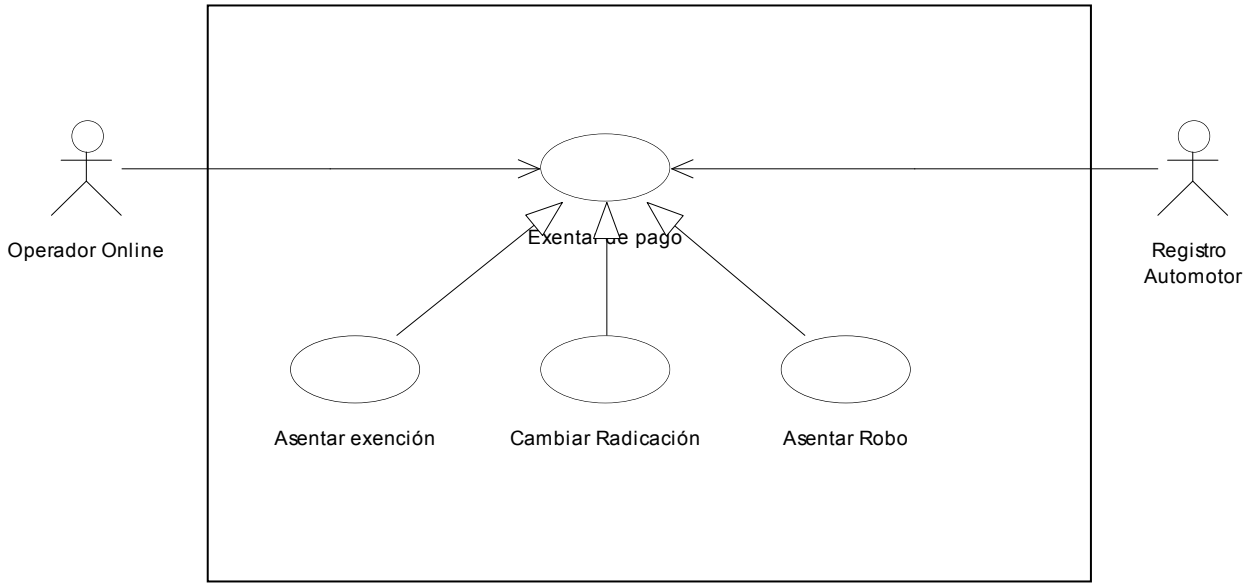


Diagrama de Exención de Pago - Sección Secundaria
Figura 3.2

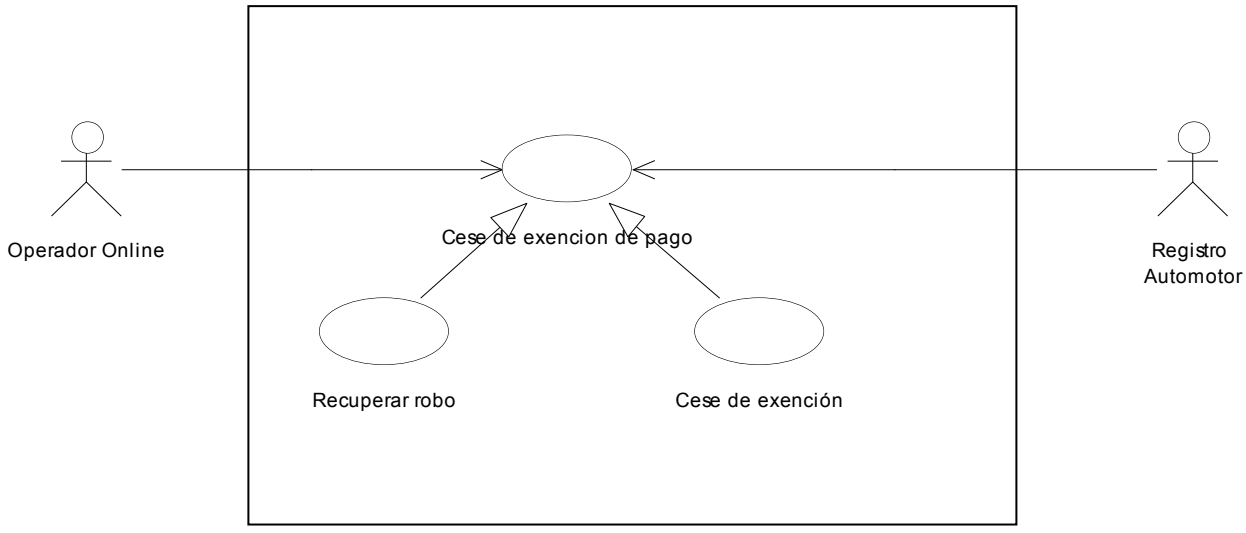


Diagrama de Cese de Exención de Pago - Sección Secundaria
Figura 3.3

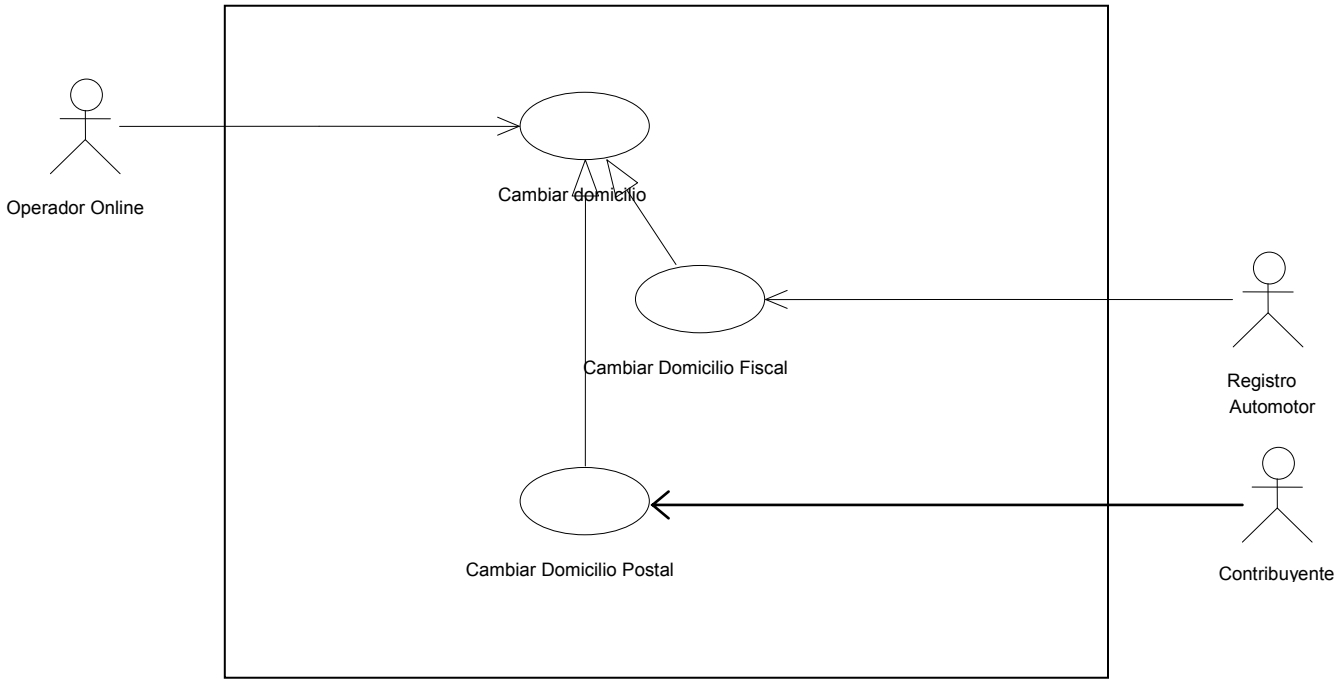


Diagrama de cambio de domicilio - Sección Secundaria
Figura 3.4

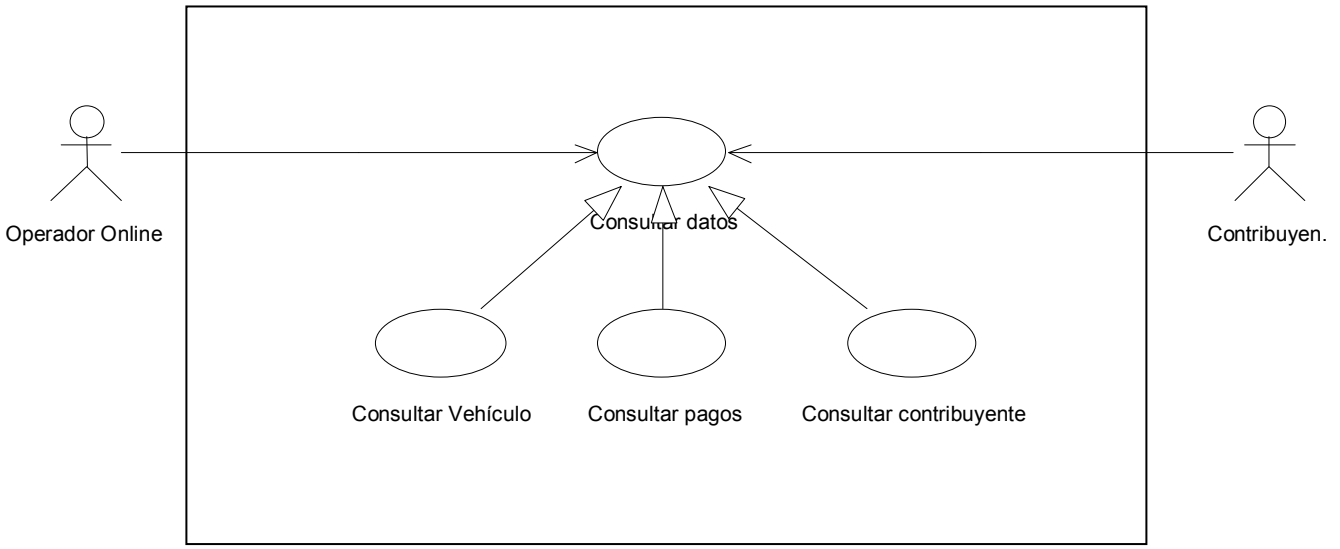


Diagrama de consulta - Sección Secundaria
Figura 3.5

3.8- Descripción de los casos de uso en formato extendido

3.8.1- Casos de uso seleccionados para desarrollar en el primer ciclo de desarrollo

Elegiremos los siguientes casos de uso para describir en forma extendida:

Pagar Impuesto
Liquidar impuesto
Imputar impuesto

Estos casos de uso constituyen el "Subsistema de Liquidación e imputación de pagos".

3.8.2 - ¿Por qué elegimos estos casos de uso?

Consideremos los siguientes aspectos:

- *La no resolución de estos casos de uso, implica el fracaso de la aplicación. Si no, habría que preguntarse ¿Cómo podría funcionar una aplicación de control de pago de un impuesto sin un buen subsistema de liquidación y de imputación?.*

- *La metodología del proceso unificado de desarrollo de software, nos indica que debemos resolver primero lo más difícil. Siguiendo con este planteo; consideramos que, los casos de uso mencionados, encierran gran variedad de conceptos y decisiones y por ende, poseen gran grado de complejidad y de dificultad.*

- *Si solucionamos estos casos de uso, habremos adquirido una gran variedad de conocimientos sobre el sistema, que nos ayudarán al resolver el resto de la aplicación.*

- *Son casos de uso muy utilizados por los actores y en consecuencia, unos de los primeros en ser solicitados en el tiempo.*

Por todas estas causas, consideramos que los casos de uso mencionados son ampliamente significativos y siguiendo con la metodología del proceso unificado de desarrollo, los resolveremos primero.

3.8.3- Descripción del caso de uso Liquidar Impuesto

Caso de uso: *Liquidar Impuesto*
Actores: *Contribuyente(iniciador), Operador*
Propósito: *Realizar una liquidación*
Descripción: *El contribuyente arriba a la oficina de operación para solicitar una liquidación para posterior pago. El operador registra los años cuotas que el contribuyente desea liquidar y luego asienta los datos de la liquidación. El contribuyente se retira con la liquidación en mano para realizar el pago en algún banco.*

Tipo: *Primario y esencial*
Referencias cruzadas: *Funciones: 1.1, 1.2, 1.3, 1.4, 1.6, 1.8*
Caso de uso: el operador debe tener completado el caso de uso "Log In".

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el contribuyente arriba a la oficina de operación y le informa al operador la patente que desea liquidar	
2-El operador marca los años – cuotas que el contribuyente desea se le liquiden.	*3-Determina el precio del impuesto para cada año – cuota.*
4-Cuando el operador termina de marcar todos los años cuotas que se liquidarán para ese vehículo, indica al sistema el fin de la selección.	*5-Calcula y presenta el total de la liquidación*
	6- Se imprime la liquidación de los períodos seleccionados.
7-El operador le entrega la liquidación al contribuyente quien se retira con su liquidación en mano.	

Cursos alternativos

1 - El código de patente es invalida o inexistente. Se indica con error.

3.8.4- Descripción del caso de uso Pagar Impuesto

SECCION PRINCIPAL

Caso de uso: *Pagar Impuesto*
Actores: *Contribuyente(iniciador), Operador*
Propósito: *Captura una liquidación y un pago del Impuesto*
Descripción: *El contribuyente arriba a la oficina de operación para realizar el pago del Impuesto. El operador registra los años cuotas que el contribuyente desea pagar y luego se asienta el pago, previa autorización del mismo. El contribuyente se retira con el comprobante de pago en mano.*

Tipo: *Primario y esencial*
Refer. cruzadas: *Funciones: 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.8, 1.9, 2.1, 2.2, 2.3, 2.4.*
Caso de uso: el operador debe tener completado el caso de uso "Log In".

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el contribuyente arriba a la oficina de operación y le informa al operador identificación del vehículo.	
2-El operador marca los años – cuotas que el contribuyente desea pagar.	
	3-Determina el precio del impuesto para cada año – cuota.
4-Cuando el operador termina de marcar todos los años cuotas que se liquidarán para ese vehículo, indica al sistema el fin de la selección.	
	5-Calcula y presenta el total de la liquidación
6-El operador informa al contribuyente sobre el importe a pagar	
7-El contribuyente selecciona el tipo de pago: - *Si el pago es por tarjeta de crédito, ver sección "Pago por tarjeta de crédito"* - *Si el pago es por tarjeta de cuenta bancaria, ver sección "Pago por Tarjeta de cuenta bancaria"*	
	8-Logs del pago realizado
	9-Actualización para asentar el pago al vehículo
	10- Generación de comprobante de pago
11-El operador le entrega al contribuyente el comprobante de pago	
12-El contribuyente se retira de la oficina con el comprobante de pago en mano	

Cursos alternativos

- *1 - El código de patente es invalida o inexistente. Se indica con error.*
  El vehículo no posee deuda. Se informa al contribuyente
- *7 - El contribuyente no puede pagar y cancela la operación de "liquidación y pago"*

La mayoría de las veces, el contribuyente no sabe que años cuotas debe y por ello es conveniente mostrar (previa a la liquidación) una pantalla con los años cuotas adeudados, el monto emitido, pagado y adeudado. Sin embargo postergaremos el tratamiento de esta función que debería estar incluida en este caso de uso para una próxima iteración. Esta decisión se debe a que no queremos complejizar demasiado este caso de uso en esta etapa de desarrollo.

SECCION: Pago por tarjeta de crédito

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-El cliente comunica los datos de su tarjeta para realizar el pago.	*2-Generación de un requerimiento de pago por tarjeta.*
	3-Envío del requerimiento al servicio de Autorización de tarjeta.
4-El Servicio de Autorización de Tarjeta autoriza el pago	
	5- Recepción de la aprobación de la tarjeta de crédito desde el Servicio de Autorización de Tarjeta.
	6-El puesto registra el pago de tarjeta y asienta la información de aprobación de la tarjeta en el sistema de Cuenta de Entrantes (se debe asentar el dinero que la tarjeta le debe a la entidad recaudadora).
	7-Muestra un mensaje de autorización de pago por tarjeta.

Cursos alternativos

- *4 – El Servicio de Autorización de Crédito no autoriza el monto de pago. Se sugieren distintos métodos de pago.*

SECCION: Pago por tarjeta de cuenta bancaria

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-El cliente comunica la información de su cuenta bancaria para el pago del impuesto.	*2-Generación de un requerimiento de pago por cuenta bancaria.*
	3-Envío del requerimiento al Servicio de Autorización del Banco.
4-El Servicio de Autorización de Cuenta Bancaria autoriza el pago	
	5- Recepción de la aprobación del pago por cuenta desde el Servicio de Autorización del Banco.
	6-El puesto registra el pago a través de cuenta y asienta la información de aprobación al sistema de Cuenta de Entrantes (se debe asentar el dinero que el banco le debe a la entidad recaudadora).
	7-Muestra un mensaje de autorización de pago por cuenta bancaria.

Cursos alternativos

- *4 – El Servicio de Autorización de Banco no autoriza el monto de pago. Se sugieren distintos métodos de pago.*

3.8.5- Descripción del caso de uso Imputar Impuesto

Caso de uso:	*Imputar pago*
Actores:	*Entidad Bancaria(iniciador), Operador*
Propósito:	*Realizar asentamiento de pagos*
Descripción:	*La entidad bancaria suministra la información necesaria para el asentamiento de pagos realizados en su institución. El operador toma la información, e imputa los pagos informados.*
Tipo:	*Primario y esencial*
Referencias cruzadas:	*Funciones: 1.5, 1.8, 2.5, 2.6*

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando la entidad bancaria envía información sobre los pagos realizados en su institución.	
2-El operador ingresa los datos al sistema.	
	3-Valida la información ingresada.
	4-Imputa los pagos que correspondan.
	5-Asienta la información sobre el pago ingresado en el sistema Cuenta de Entrantes (se debe asentar el dinero que el banco le debe a la entidad recaudadora).
	6-Imprime totales sobre los pagos asentados y forma de imputación
7-El operador le entrega los datos de la imputación a la entidad bancaria (dinero imputado, dominio a los que se le imputaron pagos, etc.)	

Cursos alternativos

3- En la validación de los datos resulta que son incorrectos. No se produce la imputación y se devuelven los mismos a la entidad bancaria para su corrección.

3.9- Asunciones realizadas en el primer ciclo de desarrollo de la aplicación

Debido a que nos encontramos en el primer ciclo de desarrollo y para no complejizar en demasía los requerimientos a abordar, haremos las siguientes asunciones:

- *El operador no tiene que hacer login.*
- *No se hace mantenimiento de inventario de los pagos (dinero entrante y saliente de la institución) sino que se verá como se asientan los pagos para un contribuyente dado.*
- *Los períodos adeudados son conocidos por el operador.*
- *En una hoja de liquidación pueden entrar todos los períodos que desee solicitar el contribuyente.*
- *En una hoja de comprobante de pago entran todos los períodos pagados por el contribuyente.*
- *No se tratarán las formas de pago, asumiremos que las tarjetas están siempre autorizadas y en consecuencia siempre se imputará el pago.*
- *Todas las liquidaciones poseen un único formato (igual vencimiento, fecha de barra, etc.)*

3.10- Descripción de los casos de uso en forma extendida. Versión 2

Antes de pasar al modelo conceptual, creemos conveniente expandir un poco más los casos de uso de la sección anterior. El motivo es que una vez que terminamos de realizar la descripción, leímos el curso típico de eventos y nos dimos cuenta, que había partes en donde se enunciaban los procesos a realizar pero no se explicaba lo que realmente se hacía en ese proceso (por ejemplo en imputación de pagos decimos que hay que validar el pago, pero no explicamos como se hace). A continuación mostraremos una segunda versión de la descripción de los casos de uso que apunta a solucionar el problema antes planteado.

Nota: Se remarcarán aquellos textos que fueron modificados.

3.10.1- Descripción en forma extendida del caso de uso Liquidar Impuesto – Versión 2

Caso de uso:	*Liquidar Impuesto*
Actores:	*Contribuyente(iniciador), Operador*
Propósito:	*Realizar una liquidación*
Descripción:	*El contribuyente arriba a la oficina de operación para solicitar una liquidación para posterior pago. El operador registra los años cuotas que el contribuyente desea liquidar y luego asienta los datos de la liquidación. El contribuyente se retira con la liquidación en mano para realizar el pago en algún banco.*
Tipo:	*Primario y esencial*
Referencias cruzadas:	*Funciones: 1.1, 1.2, 1.3, 1.4, 1.6, 1.8*

Caso de uso: el operador debe tener completado el caso de uso "Log In".

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el contribuyente arriba a la oficina de operación y da al operador la identificación del vehículo que desea liquidar	
2-El operador marca los años – cuotas que el contribuyente desea se le liquiden.	
	3-Determina la deuda actualizada de cada año – cuota de ese vehículo de acuerdo a su fecha de vencimiento, fecha de vencimiento de la liquidación, importe emitido, importe pagado, deuda original, índice de actualización.
	4- Obtenido dicho valor lo suma al total de la liquidación.
5-Cuando el operador termina de marcar todos los años cuotas que se liquidarán para ese vehículo, indica al sistema el fin de la selección.	
	6- Calcula los demás datos de la liquidación: fecha de vencimiento de la liquidación, fondo educativo, datos del vehículo, barras de liquidación.
	7- Se imprime la liquidación de los períodos adeudados.
8-El operador le entrega la liquidación al contribuyente quien se retira con su liquidación en mano.	

Cursos alternativos

1 - El código de patente es invalida o inexistente. Se indica con error.

3.10.2 – Descripción en forma extendida del caso de uso Pagar Impuesto. Versión 2

SECCION PRINCIPAL

Caso de uso: *Pagar Impuesto*
Actores: *Contribuyente(iniciador), Operador*
Propósito: *Captura una liquidación y un pago del Impuesto*

Descripción: *El contribuyente arriba a la oficina de operación para realizar el pago del Impuesto. El operador registra los años cuotas que el contribuyente desea pagar y luego se asienta el pago, previa autorización del mismo. El contribuyente se retira con el comprobante de pago en mano.*

Tipo: *Primario y esencial*

Refer. cruzadas: *Funciones:1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.8, 1.9, 2.1, 2.2, 2.3, 2.4*
Caso de uso: el operador debe tener completado el caso de uso "Log In".

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el contribuyente arriba a la oficina de operación y le informa al operador la identificación del vehículo.	
2-El operador marca los años – cuotas que el contribuyente desea pagar.	
	3-Determina la deuda actualizada de cada período marcado a pagar según el importe emitido, importe pagado, deuda original, coeficiente de liquidación, fecha de vencimiento original, fecha de vencimiento de la liquidación.
	4- Suma la deuda actualizada al total a pagar.
5-Cuando el operador termina de marcar todos los años cuotas que desea pagar para ese vehículo, indica al sistema el fin de la selección.	
	6- Calcula el total a pagar.
7- El operador informa al contribuyente sobre el total a pagar.	
8- El contribuyente selecciona el tipo de pago:	

- *Si el pago es por tarjeta de crédito, ver sección "Pago por tarjeta de crédito".* - *Si el pago es por tarjeta de cuenta bancaria, ver sección "Pago por Tarjeta de cuenta bancaria".*	
	9- Se asienta el total a pagar en cuenta de haberes de la entidad recaudadora. ***10-Actualización de los datos del vehículo a fin de asentar el importe pagado para cada período marcado.*** *11- Generación de comprobante de pago.*
12-El operador le entrega al contribuyente el comprobante de pago. *13-El contribuyente se retira de la oficina con el comprobante de pago en mano.*	

SECCION: Pago por tarjeta de crédito

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-El cliente comunica los datos de su tarjeta para realizar el pago.	*2-Generación de un requerimiento de pago por tarjeta.* *3-Envío del requerimiento al servicio de Autorización de tarjeta.*
4-El Servicio de Autorización de Tarjeta autoriza el pago.	*5- Recepción de la aprobación de la tarjeta de crédito desde el Servicio de Autorización de Tarjeta.* *6-El puesto registra el pago de tarjeta y asienta la información de aprobación de la tarjeta en el sistema de Cuenta de Entrantes (se debe asentar el dinero que la tarjeta le debe a la entidad recaudadora).* *7-Muestra un mensaje de autorización de pago por tarjeta.*

Cursos alternativos

4- El Servicio de Autorización de Crédito no autoriza el monto de pago. Se sugieren distintos métodos de pago.

SECCION: Pago por tarjeta de cuenta bancaria

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-El cliente comunica la información de su cuenta bancaria para el pago del impuesto.	*2-Generación de un requerimiento de pago por cuenta bancaria.*
	3-Envío del requerimiento al Servicio de Autorización del Banco.
4-El Servicio de Autorización de Cuenta Bancaria autoriza el pago	*5- Recepción de la aprobación del pago por cuenta desde el Servicio de Autorización del Banco.*
	6-El puesto registra el pago a través de cuenta y asienta la información de aprobación al sistema de Cuenta de Entrantes (se debe asentar el dinero que el banco le debe a la entidad recaudadora).
	7-Muestra un mensaje de autorización de pago por cuenta bancaria.

Cursos alternativos

4- El Servicio de Autorización de Banco no autoriza el monto de pago. Se sugieren distintos métodos de pago.

3.10.3 – Descripción en forma extendida del caso de uso Imputar Impuesto. Versión 2

Caso de uso: *Imputar pago*
Actores: *Entidad Bancaria(iniciador), Implementador*

Propósito: *Realizar asentamiento de pagos*

Descripción: *La entidad bancaria suministra la información necesaria para el asentamiento de pagos realizados en su institución. El operador toma la información, e imputa los pagos informados.*

Tipo: *Primario y esencial*

Referencias cruzadas: *Funciones: 1.5, 1.8, 2.5, 2.6*

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando la entidad bancaria envía los datos de pagos realizados en su intitución.	
2-El implementador del pedido ejecuta el programa de imputación.	***3- Se realiza la validación de cada pago ingresado:*** ***3.1-comparando los importes emitidos de los períodos a imputar respecto a los importes emitidos que fue guardado en los datos de la liquidación*** ***3.2- Reconstruyendo el supuesto cobrado con el coeficiente de actualizacion (desde la fecha de barra a la fecha de pago) y el importe de la barra y verificando su igualdad con el importe cobrado en los datos del pago*** ***3.3- Verificando el formato de todos los datos de pagos antes especificados***
	4-Imputa los pagos que correspondan, sumando el importe emitidos del pago al importe pagado del vehículo.
	*5-Asienta la información sobre el pago ingresado en el sistema Cuenta de Entrantes (se debe asentar el dinero que el banco le debe a la entidad recaudadora**).***
7- El implementador le entrega los datos de la imputación a la entidad bancaria (total de toda la imputación, dominios imputados)	*6-Imprime totales sobre los pagos asentados*

Curso alternativo:

3- En la validación de los datos resulta que son incorrectos. No se produce la imputación y se devuelven los mismos a la entidad bancaria para su corrección.

Capítulo 4 - Modelo Conceptual

4.1- Descripción del modelo.

Un modelo conceptual es una representación de conceptos en el dominio del problema.

A través de este modelo identificamos los conceptos u objetos significantes y por esto constituye una herramienta esencial en el análisis orientado a objetos. Es importante hacer un buen esfuerzo en obtener un buen modelo conceptual en pro del buen resultado del diseño y de la implementación.

La identificación de conceptos es parte de una investigación del dominio del problema.

La característica clave del modelo conceptual es que ***representa cosas del mundo real no componentes de software*** *(o sea que pone énfasis en los conceptos del dominio, no en las entidades de software).*

En UML, un modelo conceptual es ilustrado con un conjunto de diagramas estáticos en el cual las operaciones no son definidas. Estos diagramas muestran conceptos, asociaciones entre conceptos y atributos de conceptos.

El modelo conceptual ayuda a clarificar la terminología o vocabulario del dominio del problema al descomponer el espacio en unidades comprensibles (conceptos). Así el modelo conceptual puede verse como un modelo que comunica términos importantes y la relación entre ellos.

La descomposición (división y vencer) es una estrategia para tratar con la complejidad por división del espacio del problema en unidades comprensibles. En el análisis algorítmico la descomposición es por proceso o funciones. En cambio en nuestro caso, análisis orientado a objetos, la descomposición es fundamentalmente por conceptos. Esta es una distinción esencial entre el análisis orientado a objetos y el análisis algorítmico: la división por conceptos (objetos) y la división por funciones.

4.1.1- ¿Para qué realizamos el modelo conceptual?

- *Sirve para descomponer el espacio del problema en unidades comprensibles (descomponer en conceptos).*
- *Ayuda a clarificar la terminología o vocabulario del dominio del problema.*
- *Puede ser visto como un modelo que comunica (tanto a las partes interesadas como a los desarrolladores) conceptos importantes y como se relacionan.*

4.1.2- Pasos a seguir para realizar el modelo conceptual:

1- *Listar los conceptos candidatos usando la Lista de Categoría de Conceptos y la identificación de los sustantivos en las frases del curso típico de eventos del caso de uso.*
2- *Dibujar los conceptos en el modelo conceptual*
3- *Sumar las asociaciones necesarias para registrar relaciones entre conceptos.*
4- *Sumar los atributos necesarios para completar los requerimientos de información.*

4.2- Acerca de la organización del capítulo.

A continuación realizaremos el modelo de conceptual que incluye las funcionalidades de los tres casos de uso ya desarrollados, a saber:

1- *Liquidar Impuesto*
2- *Pagar Impuesto*
3- *Imputar Pago*

Primero realizaremos el modelo conceptual del caso de uso "LiquidarImpuesto", y luego lo iremos completando para incorporar los otros 2 casos de uso ("Pagar impuesto" y "Imputar impuesto")

4.3- Modelo conceptual para el caso de uso "Liquidar Impuesto".

4.3.1- Encontrando conceptos a través de lista de Categoría de conceptos.

Categoría de Conceptos	
Objetos tangibles o físicos	*Terminal de operación*
Especificación, diseño o descripción de cosas	*Especificación del vehículo*
Lugares	*Oficina de operación*
Transacciones	*Liquidación*
Items de la línea de transacción	*Línea períodos (años-cuotas) liquidados*
Roles de personas	*Operador on-line*
Contenedor de otras cosas	*Oficinas* *Bases de Datos*
Cosas que contiene	*Especificación de vehículos* *Identificación vehículo* *Período vehículo* *Período emitidos* *Importe emitido* *Importe pagado* *Fecha de vencimiento*
Otros computadores o sistemas mecánicos externos a nuestro sistema	*Generador de barras*
Conceptos no abstractos	*Deuda actualizada* *Indice de liquidación* *Fecha de vencimiento de la liquidación* *Total de liquidación* *Fondo educativo*
Organizaciones	*Bancos – Ente Recaudador*
Eventos	*Liquidación*
Procesos	*Liquidando períodos*
Reglas y pólizas	
Catálogos	*Catálogo de vehículos*
Registros de finanzas, trabajos, contratos y materias legales	
Instrumentos financieros y servicios	
Manuales y libros	*Manual del operador*

4.3.2- Encontrando conceptos a través de la identificación de sustantivos en las frases del curso típico de eventos del caso de uso.

Subrayaremos los sustantivos del curso típico de eventos de la versión 2 del caso de uso Liquidar impuesto.

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el contribuyente arriba a la oficina de operación y da al operador la identificación del vehículo que desea liquidar	
2-El operador marca los años – cuotas que el contribuyente desea se le liquiden.	*3-Determina la deuda actualizada de cada año – cuota de ese vehículo de acuerdo a su fecha de vencimiento, fecha de vencimiento de la liquidación, importe emitido, importe pagado, deuda original, índice de actualización*
	4- Obtenido dicho valor lo suma al total de la liquidación
5-Cuando el operador termina de marcar todos los años cuotas que se liquidarán para ese vehículo, indica al sistema el fin de la selección.	*6- Calcula los demás datos de la liquidación: fecha de vencimiento de la liquidación, fondo educativo, datos del vehículo, barras de liquidación.*
	7- Se imprime la liquidación de los períodos adeudados.
8-El operador le entrega la boleta de liquidación al contribuyente quien se retira con su liquidación en mano.	

En base a los sustantivos subrayados obtenemos los siguientes conceptos candidatos:

oficina de operación	*operador*
identificación del vehículo	*períodos emitido*
vehículo	*liquidación*
deuda actualizada	*fecha de vencimiento*
vencimiento de la liquidación	*importe emitido*
importe pagado	*deuda original*
índice de actualización	*total de la liquidación*
fondo educativo	*datos del vehículo*
barras de liquidación	*boleta de liquidación*
línea periodo liquidado	

Aclaración:

Cuando en el punto 2 del curso típico de eventos decimos:
"El operador marca los años – cuotas que el contribuyente desea se le liquiden"

Estamos hablando del año-cuota (o período) marcado, es el período que se va a liquidar o el período que se encuentra en la línea de periodos a liquidar.

En cambio, cuando en el punto 3 decimos:
"Determina la deuda actualizada de cada año – cuota de ese vehículo de acuerdo a su fecha de vencimiento, fecha de vencimiento de la liquidación, importe emitido, importe pagado, deuda original, índice de actualización"

Aquí estamos hablando del período del vehículo (el vehículo contendrá todos los períodos para los cuales el contribuyente tiene la obligación impositiva, los comprendidos entre la fecha de alta y el año de proceso)

*La **Especificación de Vehículo**, **Datos del vehículo** o **Vehículo** constituyen el mismo concepto; ya que siempre nos estamos refiriendo a las características de un vehículo dado. Por esto solo lo pusimos una vez con el nombre de **Vehículo**.*

4.3.3- Conceptos que no serán incluidos en el modelo conceptual.

*La **boleta de liquidación** es un reporte que es entregado al contribuyente para que lo lleve al banco, para que éste pueda proceder al cobro del impuesto. No mostraremos este concepto puesto que todo los datos que posee la boleta son derivables de otras fuentes.*

*El **manual del usuario** tampoco lo incluiremos como un concepto dado que constituye una característica irrelevante para comprender el dominio del problema.*

4.3.4- Encontrando asociaciones a través de la lista de asociaciones más comunes.

A es parte física de B	*----------*
A es una parte lógica de B	*Períodos liquidado / Línea de período liquidado* *Fecha vcto original / Línea de período liquidado* *Indice de liquidación / Línea de período liquidado* *Deuda original / Línea de período liquidado* *Deuda Actualizada / Línea de período liquidado* *Barra / Liquidación* *Identificación del vehículo / Liquidación* *Línea de período liquidado / Liquidación* *Fondo Educativo / Liquidación* *Total Abonar / Liquidación*
A está físicamente contenida en o sobre B	*Terminal / Oficina*
A está lógicamente contenida en B	*Vehículo / Catálogo de vehículo* *Catálogo vehículo / Oficina* *Identificación del vehículo / Vehículo* *Período Vehículo / Vehículo* *Período emitido / Período Vehículo* *Importe Emitido / Período Vehículo* *Importe Pagado / Período Vehículo* *Fecha vcto original / Período Vehículo*
A es descripción de B	
A es una línea de items de una transacción o reporte de B	*Período liquidado / Liquidación*
A es conocido / logeado / reportado / capturado en B	*Liquidación (completada en) Oficina* *Liquidación (capturada en) Terminal* *Deuda Original (calculada con) Importe emitido*

	Deuda Original (calculada con) Importe pagado *Deuda Actualizada (calculada con) Coeficiente de actualización* *Deuda Actualizada (calculada con) Deuda original* *Coeficiente de Actualización (calculada con)* *Fecha de vencimiento Original* *Coeficiente de Actualización (calculada con)* *Fecha de vencimiento Liquidación* *Barras (generadas por) Generador de barras*
A es miembro de B	*Operador / Oficina de liquidación*
A es una subunidad organizacional de B	----------
A usa o maneja a B	*Operador / Terminal*
A comunica con B	*Operador / Contribuyente*
A está relacionado a la transacción de B	*Contribuyente / Liquidación* *Operador / Liquidación*
A es una transacción relacionada a otra transacción de B	
A es el próximo a B	
A es poseída por B	*Terminal / Oficina*

4.3.5- Dibujando la primer y segunda versión del modelo conceptual.

En base a lo desarrollado de las 3 últimas secciones obtenemos la primer versión del modelo conceptual.

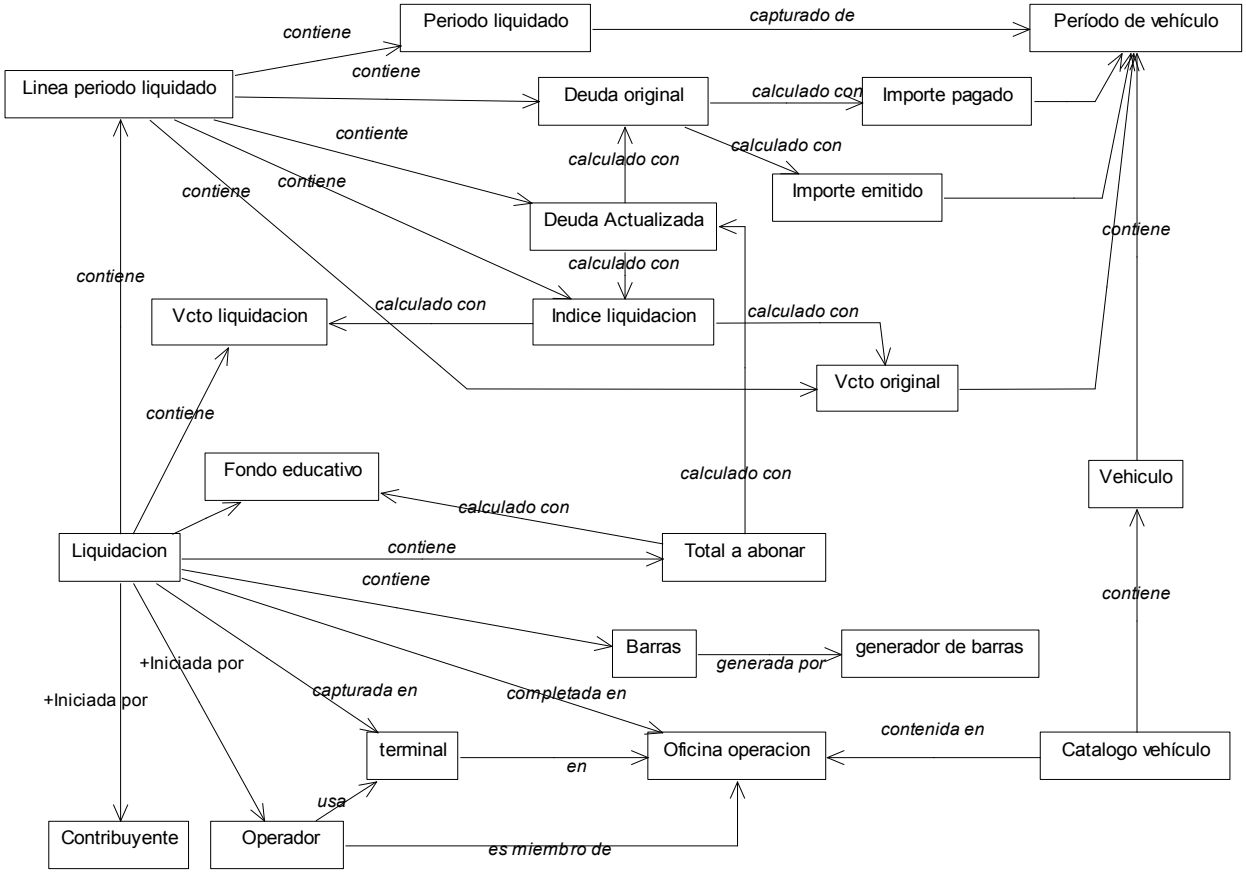


Figura 4.1
Diagrama de Modelo Conceptual – Versión 1

Observando la figura 4.1, vemos que hemos considerado como conceptos, cosas que en realidad son atributos, a saber:

- *Total a abonar: es un número que se calcula sumando todos los importes actualizados adeudados de la presente liquidación y el fondo educativo. Es por ende más que un concepto un atributo de la liquidación.*
- *Fondo Educativo: es un número calculable en el momento de la liquidación dependiendo de la valuación del vehículo y lo pondremos como atributo de la liquidación(pues la valuación de un vehículo varia en el tiempo)*
- *Vencimiento de la liquidación: es una fecha determinada por pautas dadas por el Ente Recaudador (en este momento 5 días a partir del día que se saco la liquidación), es dato elemental calculable en el momento de la liquidación y la pondremos como atributo de ella.*
- *Periodo liquidado: son parte de la línea de periodo liquidado y constituye un dato elemental que lo pondremos como concepto de dicha línea.*
- *Fecha de vencimiento Original: Fecha de vencimiento del período liquidado en el momento de la liquidación.*
- *Indice de actualización: calculado en base a la fecha de vencimiento de la liquidación y la fecha de vencimiento original del período liquidado. Es calculado en la línea de liquidación y lo pondremos como atributo de ese concepto.*
- *Deuda original: se calcula restando el importe emitido y el importe pagado (atributos de vehículo). Debido de que es parte de la línea de liquidación y que puede variar en el tiempo, lo pondremos como atributo de ella*
- *Deuda actualizada: esta contenida en la línea de periodo liquidado y se obtiene multiplicando la deuda original por el índice de actualización. Lo pondremos como atributo de la línea período liquidado.*
- *Vencimiento original: identifica la fecha en que vence determinado período de un vehículo, constituye un dato de dicho concepto y por ende, lo pondremos como atributo de período de vehículo.*
- *Identificación del dominio: conjunto de caracteres conocidos por el contribuyente que identifican unívocamente a su vehículo, por ende lo pondremos como atributo de vehículo.*
- *Importe emitido: importe emitido para un período(sin índice de actualización), lo pondremos como atributo del período de vehículo*
- *Importe pagado: importe pagado por el contribuyente para un período, lo pondremos como atributo del período del vehículo.*

Nuestro diagrama conceptual queda con los siguientes conceptos candidatos:

Liquidación	*Línea Período liquidado*
Contribuyente	*Operador*
Terminal	*Barras*
Generador de barras	*Oficina de operación*
Vehículo	*Catálogo de vehículo*
Período vehículo	

A continuación reveemos las relaciones entre dichos conceptos:

A es parte física de B	
A es una parte lógica de B	*Barra / Liquidación* *Línea de período liquidado / Liquidación*
A está físicamente contenida en o sobre B	*Terminal / Oficina*
A está lógicamente contenida en B	*Vehículo / Catálogo de vehículo* *Catálogo vehículo / Oficina* *Período vehículo / Vehículo*
A es descripción de B	
A es una línea de items de una transacción o reporte de B	*Período liquidado / Liquidación*
A es conocido / logeado / reportado / capturado en / capturado de B	*Liquidación (completada en) Oficina* *Liquidación (capturada en) Terminal* *Barras (generadas por) Generador de barras* *Línea de períodos liquidados / período de Vehículo*
A es miembro de B	*Operador / Oficina de liquidación*
A es una subunidad organizacional de B	
A usa o maneja a B	*Operador / Terminal*
A comunica con B	*Operador / Contribuyente*
A está relacionado a la transacción de B	*Contribuyente / Liquidación* *Operador / Liquidación*
A es una transacción relacionada a otra transacción de B	
A es el próximo a B	
A es poseída por B	*Terminal / Oficina*

Teniendo en cuenta todo lo expuesto, la figura 4.2 se muestra segunda versión del modelo conceptual para el susbistema de liquidación del impuesto.

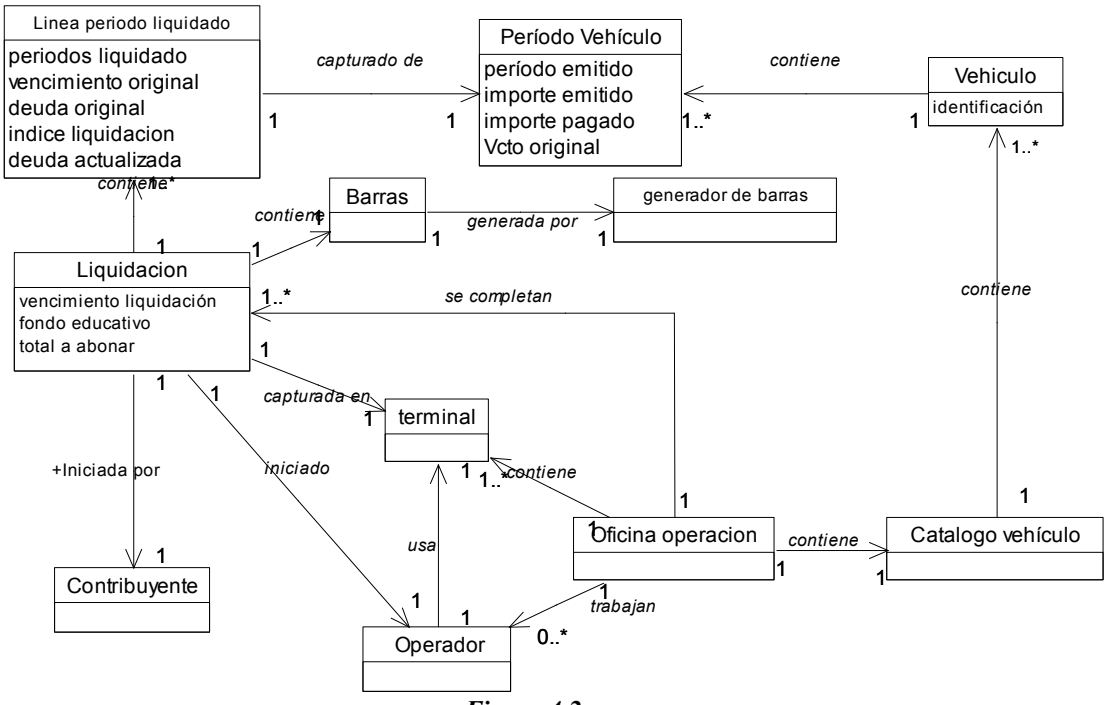


Figura 4.2
Diagrama de Modelo Conceptual – Versión 2

4.4- Incorporando al modelo conceptual el caso de uso "Pagar Impuesto"

4.4.1- Encontrando conceptos a través de lista de Categoría de conceptos.

Categoría de Conceptos	
Objetos tangibles o físicos	*Terminal de operación*
Especificación, diseño o descripción de cosas	*Especificación del vehículo*
Lugares	*Oficina de operación*
Transacciones	*Pago* *Liquidación (ver nota)*
Items de la línea de transacción	*Línea períodos liquidados (años-cuotas)*
Roles de personas	*Operador on-line*
Contenedor de otras cosas	*Oficinas* *Bases de Datos*
Cosas que contiene	*Catálogo de Vehículos* *Identificación vehículo* *Período Vehículo* *Período emitido* *Importe emitido* *Importe pagado* *Fecha de vencimiento*
Otros computadores o sistemas mecánicos externos a nuestro sistema	
Conceptos no abstractos	*Deuda actualizada* *Indice de liquidación* *Vencimiento de la liquidación* *Total de liquidación* *Fondo educativo*
Organizaciones	*Bancos – Ente Recaudador*
Eventos	*Liquidación*
Procesos	*Liquidando períodos* *Pagando períodos*
Catálogos	*Catálogo de vehículos*
Manuales y libros	*Manual del operador*

Nota:

- Consideramos que el comprobante de pago es una liquidación sin barras y con una leyenda indicando su naturaleza (una leyenda indicando que los períodos que constan en la liquidación fueron abonados por el contribuyente en la oficina de operación)

4.4.2- Encontrando conceptos a través de la identificación de sustantivos en las frases del curso típico de eventos del caso de uso.

Subrayaremos los sustantivos del curso típico de eventos de la versión 2 del caso de uso Pagar Impuesto. En esta fase del ciclo de desarrollo, no consideraremos las secciones de forma de pago, para lograr una mayor simplicidad en el caso de uso. Más adelante en la próxima fase del ciclo de desarrollo resolveremos estas secciones.

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el contribuyente arriba a la oficina de operación y le informa al operador la identificación de vehículo que desea pagar	
2-El operador marca los años – cuotas que el contribuyente desea pagar.	*3-Determina la deuda actualizada de cada período marcado a pagar según el importe emitido, importe pagado, deuda original, coeficiente de liquidación, fecha de vencimiento original, fecha de vencimiento de la liquidación*
	4- Suma la deuda actualizada al total a pagar
5- Cuando el operador termina de marcar todos los años cuotas que desea pagar para ese vehículo, indica al sistema el fin de la selección.	
	6- Calcula el total a pagar
7- El operador informa al contribuyente sobre el total a pagar	
8- El contribuyente realiza el pago	
	9- Se asienta el total a pagar en cuenta de haberes de la entidad recaudadora
	10-Actualización de los datos del vehículo a fin de asentar el importe pagado para cada período marcado
	11- Generación de comprobante de pago
12-El operador le dá al contribuyente el comprobante de pago	
13-El contribuyente se retira de la oficina con el comprobante de pago en mano	

Los conceptos candidatos que resultan de la metodología de sustantivos son:

Contribuyente	*Oficina de operación*
Operador	*Períodos*
Contribuyente	*Vehículo*
Total a pagar	*Pago*
Comprobante de pago	*Deuda actualizada*
Importe emitido	*Importe pagado*

Deuda original	*Coeficiente de liquidación*
Vencimiento original	*Vencimiento de la liquidación*
Cuenta de haberes	*Datos del vehículo*
Identificación del vehículo	*Entidad recaudadora*
Comprobante de pago	*Identificación del vehículo*

*El **comprobante de pago** es un reporte que es entregado al contribuyente para que tenga una constancia del pago realizado. Su importancia radica en e que el contribuyente pueda a través de ella realizar un reclamo de imputación de los períodos que en ella consta. Por el otro lado todos los datos en este reporte son derivables de otras fuentes. Debido a que en esta fase del ciclo de desarrollo no trataremos los reclamos de pagos, no consideramos importante incluir este concepto en el modelo conceptual.*

*La **cuenta de haberes** es una cuenta que refleja la contabilidad de la ente Recaudador. Allí se guardarán los datos de los débitos y créditos de la empresa. En este ciclo no trataremos con esta funcionalidad, luego este concepto será (por ahora) descartado.*

4.4.3- Identificando los conceptos y los atributos.

Mediante las 2 metodologías de obtención de conceptos candidatos, obtuvimos la siguiente lista de conceptos candidatos:

Contribuyente	*Oficina de operación*
Operador	*Períodos liquidados*
Contribuyente	*Vehículo*
Total a pagar	*Pago*
Deuda actualizada	*Importe emitido*
Importe pagado	*Deuda original*
Coeficiente de liquidación	*Vencimiento original*
Vencimiento de la liquidación	*Cuenta de haberes*
Datos del vehículo	*Terminal*
Liquidación	*Línea período liquidado*
Periodo emitido	*Fondo Educativo*
Bancos	*Entes Recaudadores*
Catálogo de vehículos	*Período Vehículo*

Como ya explicamos anteriormente los conceptos candidatos: períodos liquidados, total a pagar, deuda actualizada, importe emitido, importe pagado, deuda original, coeficiente de liquidación, vencimiento original, vencimiento de la liquidación, período emitido y fondo educativo; constituyen atributos de otros conceptos. Además Vehículo y Datos del vehículo constituyen el mismo concepto (justificado en sección anterior), de esta manera nuestra lista queda reducida a:

Contribuyente	*Oficina de operación*
Operador	*Contribuyente*
Vehículo	*Pago*
Cuenta de haberes	*Terminal*
Liquidación	*Línea período liquidado*
Bancos	*Entes Recaudadores*

Catálogo de vehículos Período Vehículo

Nótese que en esta lista de conceptos candidatos hay muchos que ya fueron evaluados en la versión 2 del modelo conceptual, por ende "pago" es un concepto candidato nuevo.

4.4.4- Reviendo la lista de asociaciones con los nuevos conceptos.

A es parte física de B	
A es una parte lógica de B	*Barra / Liquidación* *Línea de período liquidado / Liquidación*
A está físicamente contenida en o sobre B	*Terminal / Oficina*
A está lógicamente contenida en B	*Vehículo / Catálogo de vehículo* *Catálogo vehículo / Oficina* *Período Vehículo/Vehículo*
A es descripción de B	
A es una línea de items de una transacción o reporte de B	*Período liquidado / Liquidación*
A es conocido / logeado / reportado / capturado en / capturado de B	*Liquidación (completada en) Oficina* *Pago (completado en) Oficina ** *Liquidación (capturada en) Terminal* *Pago (capturado en) Terminal ** *Barras (generadas por) Generador de barras* *LíneaPeríodoLiquidado(conoce) PeríodoVehículo* *Pago (capturado en) Vehículo * (importe pagado)*
A es miembro de B	*Operador / Oficina de liquidación*
A es una subunidad organizacional de B	
A usa o maneja a B	*Operador / Terminal*
A comunica con B	*Operador / Contribuyente*
A está relacionado a la transacción de B	*Contribuyente / Liquidación* *Contribuyente / Pago ** *Operador / Liquidación* *Operador / Pago **
A es una transacción relacionada a otra transacción de B	*Liquidación / Pago*
A es el próximo a B	
A es poseída por B	*Terminal / Oficina*

*Las asociaciones marcadas con * son las que han sido descubiertas en esta etapa.*

4.4.5- Diagrama del modelo conceptual con el Caso de Uso "Pagar Impuesto".

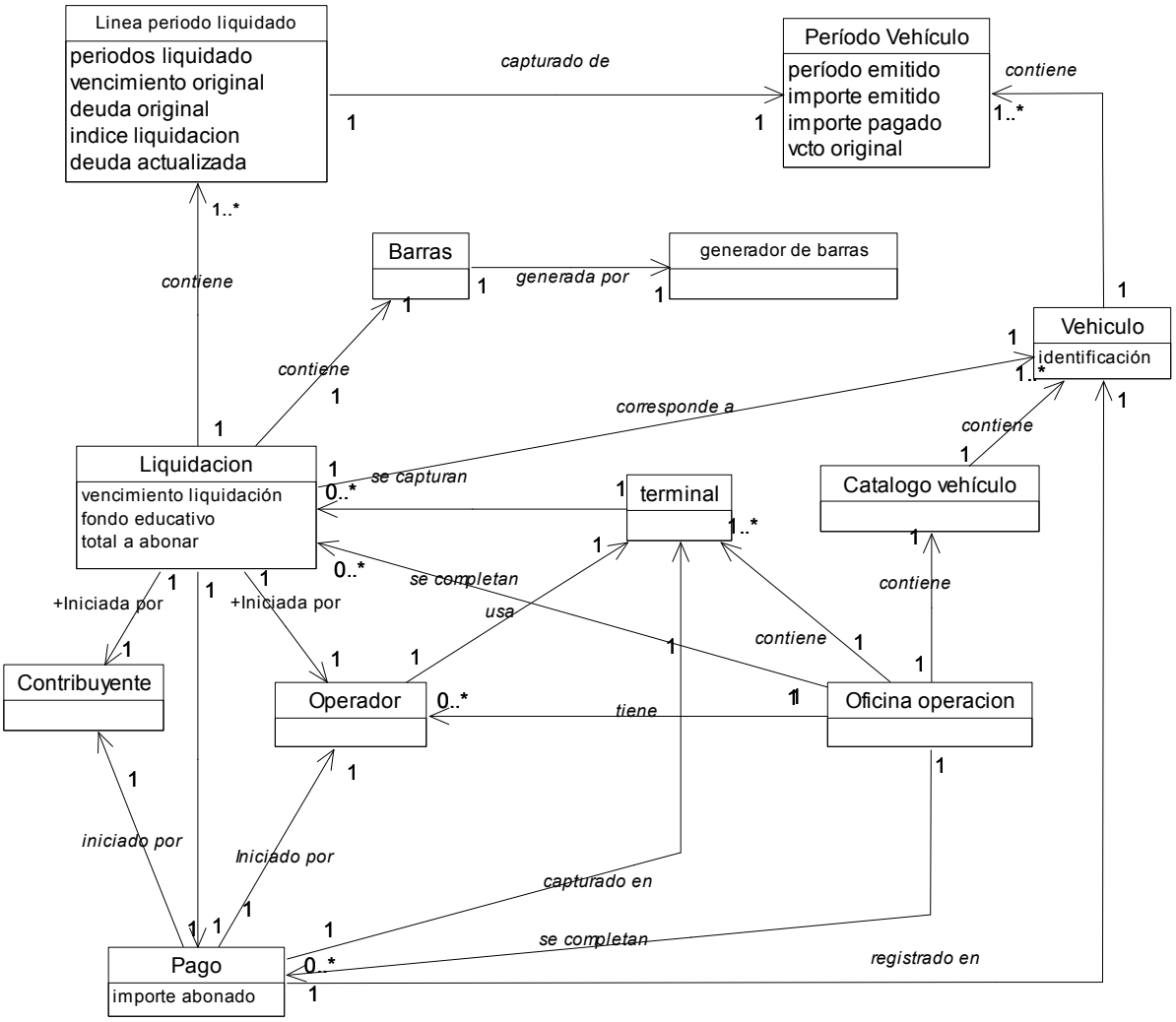


Figura 4.3
Diagrama de Modelo Conceptual – Versión 3

4.5– Incorporando al modelo conceptual el caso de uso "Imputar Pagos"

4.5.1- Encontrando conceptos a través de lista de Categoría de conceptos.

Categoría de Conceptos	
Objetos tangibles o físicos	*Terminal de operación*
Especificación, diseño o descripción de cosas	*Especificación del vehículo* *Especificación del pago* *Especificación de liquidación*
Lugares	*Oficina de implementación*
Transacciones	*Verificación* *Imputación*
Items de la línea de transacción	*Línea períodos liquidados*
Roles de personas	*Implementador del pedido*

Contenedor de otras cosas	*Oficina* *Bases de Datos*
Cosas que contiene	*Especificación de vehículos* *Especificación del pago* *Especificación de liquidaciones* *Identificación de vehículo en vehículo* *Período vehículo* *Identificación de vehículo en liquidación* *Períodos emitidos en la liquidación* *Fecha de vencimiento de la barra* *Fecha de pago* *Deuda Actualizada de Barra* *Deuda Original en la liquidación* *Identificación de vehículo en pagos* *Período pago* *Período a imputar* *Deuda Original en pagos* *Importe cobrado en pagos*
Otros computadores o sistemas mecánicos externos a nuestro sistema	*Subsistema de carga de pago*
Conceptos no abstractos	
Organizaciones	*Bancos – Ente Recaudador*
Eventos	*Verificación de pagos* *Imputación de pago*
Procesos	*Verificando períodos a imputar* *Imputando períodos*
Reglas y pólizas	
Catálogos	*Catálogo de vehículos* *Catálogo de pagos* *Catálogo de liquidaciones*
Registros de finanzas, trabajos, contratos y materias legales	
Instrumentos financieros y servicios	
Manuales y libros	*Manual del operador*

4.5.2- Encontrando conceptos a través de la identificación de sustantivos en las frases del curso típico de eventos del caso de uso.

Subrayaremos los sustantivos del curso típico de eventos de la versión 2 del caso de uso Imputar pagos.

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando la entidad bancaria envía los datos de pagos realizados en su institución.	
2-El implementador del pedido ejecuta el programa de imputación.	*3- Se realiza la validación de cada pago ingresado por el banco:*

	3.1- Comparando la deuda original de los períodos a imputar en los datos del pago respecto a la deuda original que fue guardado en los datos de la liquidación *3.2- Reconstruyendo el supuesto cobrado con el coeficiente de actualización (desde la fecha de barra a la fecha de pago) y el importe de la barra y verificando su igualdad con el importe cobrado en los datos del pago* *3.3- Verificando el formato de todos los datos de pagos antes especificados* *4-Imputa los pagos que correspondan, sumando la deuda original del pago al importe pagado del vehículo.* *5-Asienta la información sobre el pago ingresado en el sistema Cuenta de Entrantes (se debe asentar el dinero que el banco le debe a la entidad recaudadora).*
7- El implementador le da los datos de la imputación a la entidad bancaria (Total de toda la imputación, Dominios Imputados.)	*6-Imprime totales sobre los pagos asentados*

Por la metodología de sustantivos, obtenemos la siguiente lista de conceptos candidatos:

Entidad bancaria	*Datos de pagos*
Implementador del pedido	*Datos de la imputación*
Total de toda la imputación	*Dominios Imputados*
Validación del pago	*Deuda original de pago*
Períodos a imputar de pago	*Datos de la liquidación*
Supuesto cobrado	*Coeficiente de actualización*
Fecha de barra	*Fecha de pago*
Importe de la barra	*Importe cobrado*
Importe pagado del vehículo	*Pago*
Cuenta de Entrantes o haberes	*Entidad recaudadora*

4.5.3- Identificando los conceptos y los atributos.

Mediante las 2 metodologías de obtención de conceptos candidatos, obtuvimos la siguiente lista de conceptos candidatos:

Entidad bancaria	*Entidad recaudadora*
Implementador del pedido	*Cuenta de Entrantes*
Terminal	*Oficina*
Pago	*Datos de pagos*
Datos de la imputación	*Imputación*
Datos de la liquidación	*Liquidación*

Especificación de liquidaciones
Especificación del pago
Validación del pago
Imputación
Dominios Imputados
Períodos a imputar de pago
Identificación de vehículo en pago
Identificación de vehículo en liquidación
Identificación de vehículo en vehículo
Fecha de pago
Importe cobrado en pago
Período a imputar
Catálogo de liquidación
Catálogo de vehículo
Especificación del vehículo
Verificación
Total de toda la imputación
Deuda original de pago
Supuesto cobrado
Coeficiente de actualización
Fecha de barra
Importe de la barra
Deuda Actualizada de barra
Importe pagado del vehículo
Deuda Original en liquidación
Catálogo de pago
Período pago

Viendo la lista de conceptos vemos que existe duplicidad en algunos conceptos, por ejemplo:

- *Validación de pago y Verificación de pago*
- *Especificación de pagos, pagos y datos de pagos*
- *Especificación de vehículo, datos del vehículo y vehículo*
- *Datos de la liquidación, Liquidación y especificación de la liquidación*

Así nuestra lista de conceptos candidatos se reduce a los siguientes:

Entidad bancaria
Implementador del pedido
Terminal
Pago
Liquidación
Validación del pago
Total de toda la imputación
Períodos a imputar de pago
Importe cobrado en pago
Supuesto cobrado
Importe pagado del vehículo
Importe de la barra
Período emitido en liquidación
Identificación de vehículo en pago
Identificación de vehículo en liquidación
Identificación de vehículo en vehículo
Período pago
E*ntidad recaudadora*
Cuenta de Entrantes
Oficina
Imputación
Vehículo
Período a imputar
Dominios Imputados
Deuda original de pago
Fecha de pago
Coeficiente de actualización
Fecha de barra
Deuda Actualizada de barra
Deuda Original en la liquidación
Catálogo de pago
Catálogo de liquidación
Catálogo de vehículo

De estos conceptos listados arriba, resulta que:

- *Los conceptos candidatos Períodos a imputar de pago, Deuda original de pago constituye un dato elemental enviado por el banco para control de la imputación, (se compara el dato extraído por el banco en el momento del pago contra el guardado en la liquidación). Lo ubicaremos como atributo del período del pago*
- *El Importe cobrado en pago y fecha de pago, constituye respectivamente el monto que cobro el banco y la fecha de pago de la*

liquidación en el banco. Son datos elementales utilizados para la verificación previa a la imputación y los pondremos como atributos del pago.

- *El coeficiente de actualización y el supuesto cobrado son datos elementales calculados para la verificación del pago. Es por eso que constituye lo ubicaremos como atributos del concepto validación de pago.*
- *El Importe pagado del vehículo es un atributo del vehículo como ya se vio en secciones anteriores*
- *Los conceptos candidatos fecha de barra, importe de la barra, deuda actualizada de barra constituyen datos guardados en el momento de la liquidación es por eso que lo ubicaremos como atributo de dicho concepto.*
- *Con respecto a los períodos emitidos en la liquidación y deuda original de la liquidación son atributos del concepto línea periodo liquidado.*
- *Identificación de vehículo en pago: conjunto de caracteres conocidos por el contribuyente que identifican unívocamente a su vehículo. Lo pondremos como atributo del pago.*
- *Identificación de vehículo en liquidación: conjunto de caracteres conocidos por el contribuyente que identifican unívocamente a su vehículo. Lo pondremos como atributo en liquidación.*

Ahora nuestra lista queda reducida a los siguientes conceptos:

Entidad bancaria	E*ntidad recaudadora*
Implementador del pedido	*Cuenta de Entrantes*
Terminal	*Oficina*
Pago	*Imputación*
Liquidación	*Vehículo*
Validación del pago	*Período de pago*
Catálogo de liquidación	*Catálogo de pago*

*El concepto llamado **pago** introducido en este caso de uso, es distinto semánticamente hablando al que se expuso con anterioridad en el caso de uso Pagar impuesto. Efectivamente, en ese caso de uso el concepto representa el pago realizado por el contribuyente en el momento de la liquidación en la oficina de rentas, mientras que en el caso de uso Imputar pagos el concepto representa las características con que ingreso el pago de determinado dominio. Por ende, para realizar la distinción, llamaremos **pagos on-line** al concepto del caso de uso Pagar Impuesto y **pagos batch** al concepto del caso de uso Imputar Pagos.*

Por otro lado considero que Imputación y Pago batch constituyen semánticamente el mismo concepto, es por eso que eliminaremos imputación ya que realizan la misma tarea.

De esta lista los conceptos nuevos son:

Entidad bancaria	E*ntidad recaudadora*
Implementador del pedido	*Pago batch*
Validación del pago	*Período pago batch*
Catálogo de liquidación	*Catálogo de pagos*

Para poder controlar la validez de los pagos que el banco entregó para imputar, es necesario que se guarde la información acerca de la liquidación que el contribuyente fue a pagar en el banco. Los datos de todas las liquidaciones se encuentran en el catálogo de liquidación. Este concepto no fue introducido en el caso de uso Liquidar Impuesto, puesto que en ese momento no nos dimos cuenta de la necesidad de guardar datos de la liquidación. Cuando examinamos este caso de uso, vimos que la única forma de realizar un control es mediante la comparación de lo que se le liquidó al contribuyente y lo que el banco entrega para imputar. De ahí surge la necesidad de incorporación del concepto Catálogo de Liquidación y del atributo identificación de Vehículo en la Liquidación y en el PagoBatch (como forma de ubicar el par de objetos a comparar).

4.5.4- Reviendo la lista de asociaciones con los nuevos conceptos.

A es parte física de B	
A es una parte lógica de B	*Barra / Liquidación* *Línea de período liquidado / Liquidación* *Período pago batch / Pago batch **
A está físicamente contenida en o sobre B	*Terminal / Oficina*
A está lógicamente contenida en B	*Vehículo / Catálogo de vehículo* *Liquidación / Catálogo de liquidación ** *Pago batch / Catálogo de pagos ** *Período Vehículo / Vehículo* *Período pago batch / Pago batch* *Catálogo vehículo / Oficina* *Catálogo de liquidación / Oficina ** *Catálogo de pagos / Oficina **
A es descripción de B	
A es una línea de items de una transacción o reporte de B	*Período liquidado / Liquidación*
A es conocido / logeado / reportado / capturado en / capturado de B	*Liquidación (completada en) Oficina* *Pago (completado en) Oficina* *Pago batch (completada en) Oficina ** *Validación (completado en) Oficina ** *Liquidación (capturada en) Terminal* *Pago batch (capturado en) Terminal* *Barras (generadas por) Generador de barras* *LíneaPeríodoLiquidado /PeríodoVehículo* *Pago batch (capturado en) Vehículo*
A es miembro de B	*Operador / Oficina de liquidación* *Implementador / Oficina **
A es una subunidad organizacional de B	
A usa o maneja a B	*Operador / Terminal* *Implementador / Terminal **
A comunica con B	*Operador / Contribuyente* *Implementador / Entidad bancaria ** *Implementador / Ente recaudador **

A está relacionado a la transacción de B	*Contribuyente / Liquidación* *Entidad bancaria / Pago Batch ** *Ente recaudador / Pago Batch ** *Operador / Liquidación* *Pago Batch / Imputación **
A es una transacción relacionada a otra transacción de B	*Pago batch / Liquidación ** *Pago batch / Validación **
A es el próximo a B	
A es poseída por B	*Terminal / Oficina*

4.5.5- Diagrama del modelo conceptual con el Caso de Uso "Imputar Pago".

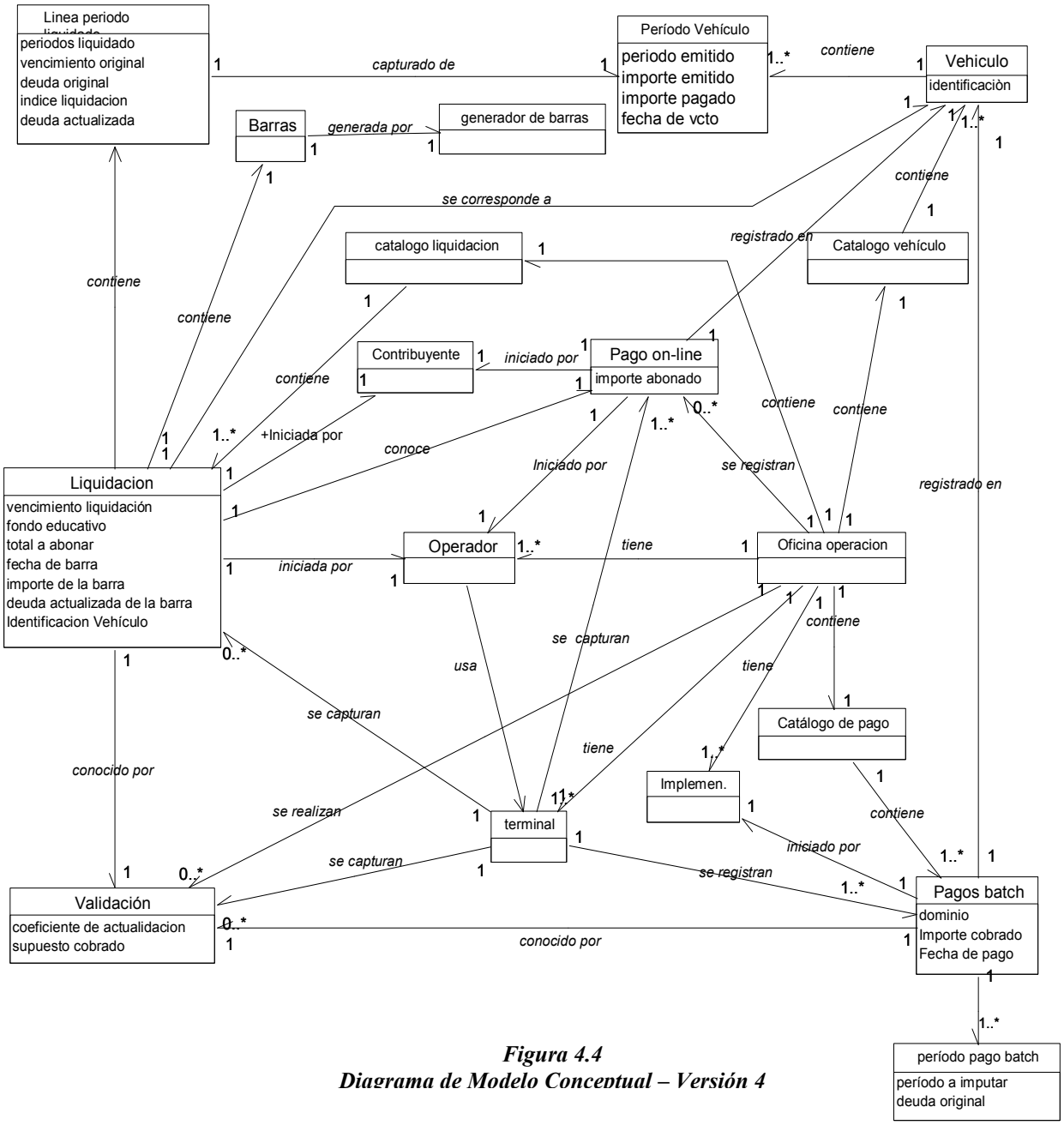


Figura 4.4
Diagrama de Modelo Conceptual – Versión 4

Capítulo 5 - Diagramas de secuencias del sistema

5.1- Descripción

Durante la realización de los casos de uso vimos como los actores interactúan sobre el sistema de software. Durante esta iteración un actor genera eventos a un sistema y requiere alguna operación en respuesta. Un diagrama de secuencias ilustra las interacciones de los actores y las operaciones inicializadas por ellos.

Un diagrama de secuencia de eventos es un cuadro que ilustra para un escenario particular de un caso de uso, los eventos que los actores externos generan, su orden y los eventos inter-sistema. Aquí todo el sistema es tratado como una caja negra, el énfasis del diagrama son los eventos que atraviesan los límites del sistema (que van de los actores al sistema).

Se debe realizar un diagrama de secuencia para el curso típico de eventos (descripto en la especificación del caso de uso) y para los cursos alternativos más interesantes.

Para un curso típico de eventos, un diagrama de secuencias muestra:

- *actores externos que interactúan directamente con el sistema*
- *el sistema (como una caja negra)*
- *los eventos del sistema que los actores generan*

*Un evento de sistema es un estímulo de entrada externa generado por un actor. Un **evento** inicializa una operación dentro del sistema. Una **operación** es la acción que se ejecuta en el sistema en respuesta a un evento.*

Tanto el evento como la operación poseen el mismo nombre, la diferencia entre ambos es conceptual (como ya se explicó antes un evento es un estímulo nombrado y la operación es la respuesta)

5.2- Acerca de la organización del capítulo

A continuación se mostrará el diagrama de secuencia para los casos de uso que estamos estudiando:

- *Liquidar Impuesto*
- *Liquidar/Pagar Impuesto*
- *Imputar Pago*

Una vez que hayamos obtenido los diagramas de secuencias, los utilizaremos para obtener las operaciones de sistemas, cuyos contratos escribiremos en el capítulo subsiguiente.

5.3- Diagrama secuencial de Liquidar Impuesto

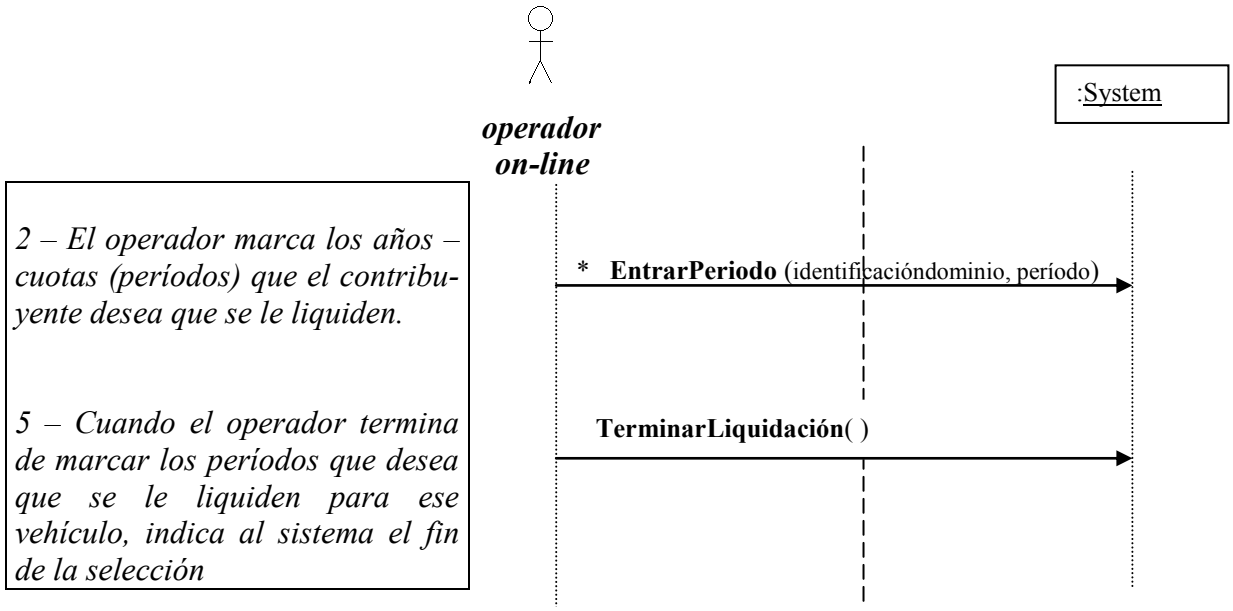


Figura 5.1

En este diagrama, hemos puesto como único actor al Operador On-line ya que el otro actor de este caso de uso (el contribuyente), no opera directamente sobre el sistema.

Por otro lado hemos puesto un asterisco delante de la operación de sistema EntrarPeríodo para indicar que se realiza 1 o más veces durante caso de uso (se ingresan 1 o más períodos a liquidar).

5.4- Diagrama secuencial de Pagar Impuesto

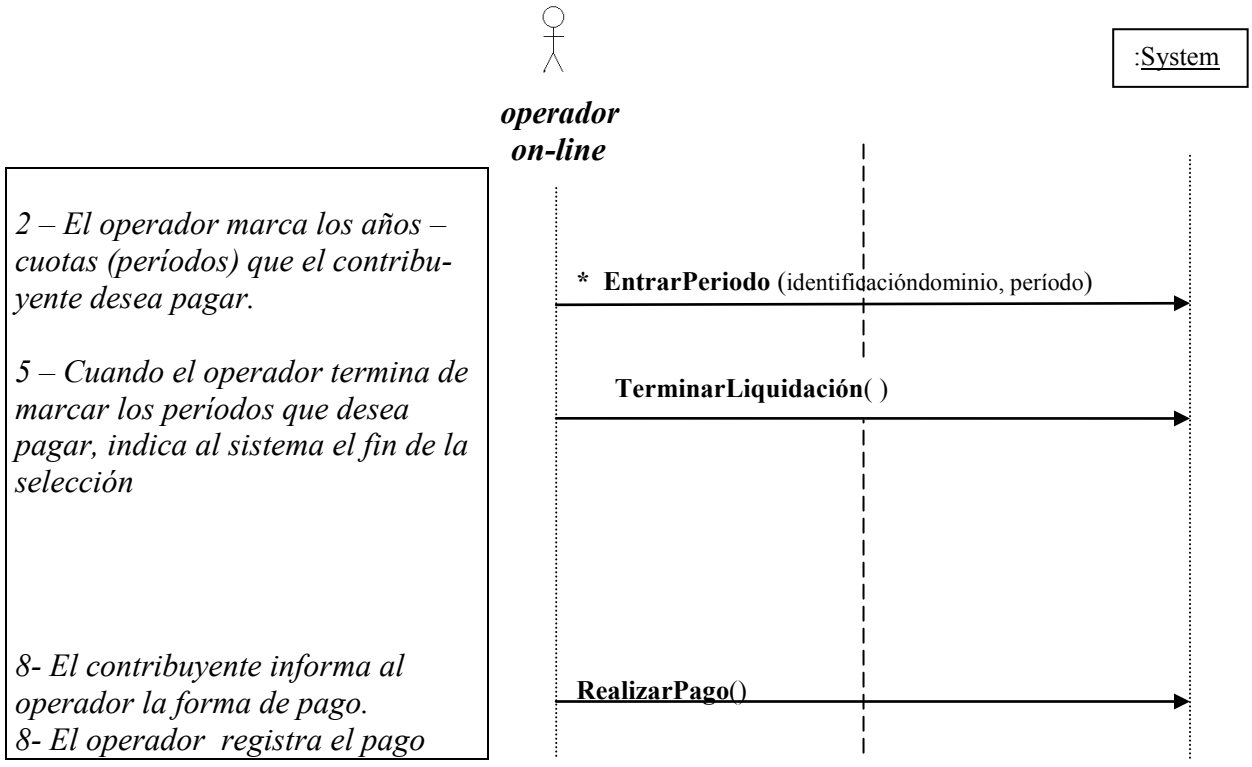


Figura 5.2

Nota:

Nuevamente en este diagrama, hemos puesto como único actor al Operador On-line (ya que el otro actor de este caso de uso, el contribuyente, no opera directamente sobre el sistema) y además hay un asterisco delante de EntrarPeríodo, para indicar que se realiza 1 o más veces.

5.5- Diagrama secuencial de Imputar Impuesto

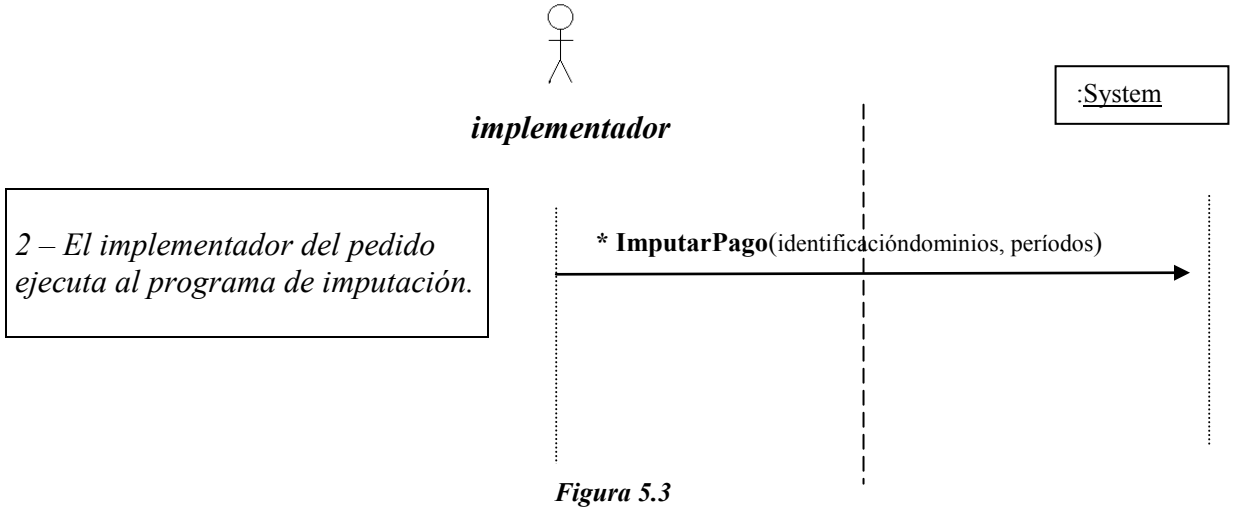


Figura 5.3

En este diagrama, hemos puesto como único actor al Implementador, ya que la entidad bancaria no constituye un actor que opere directamente con el sistema.

Por el otro lado, hemos introducido un asterisco delante de EjecutarImputación ya que una imputación puede tener uno o muchos o muchos pagos batch asociados. Cuando hablamos de pago batch, nos estamos refiriendo al pagos correspondiente a una liquidación, o sea que para cada liquidación que el contribuyente paga en el banco, habrá una nueva instancia en el catálogo de pagos Batch. Durante la imputación se deberá asociar unívocamente ese pago batch (ingresado por el banco) con la liquidación que realizó el contribuyente (cuyos datos se encuentran guardados en el catálogo de liquidación), para así asegurar su validez.

5.6- Operaciones de Sistema

Mirando las secciones anteriores de este capítulo, podemos decir ver que, para los caso de uso en estudio, existen las siguientes operaciones de sistema:

*1- **EntrarPeriodo** (identificacióndominio, período)*
*2- **TerminarLiquidación**( )*
*3- **RealizarPago**( )*
*4- **ImputarPago** (identificacióndominios, períodos)*

Capítulo 6 - Contratos

6.1- Descripción.

Los contratos ayudan a definir el comportamiento del sistema ya que describen el efecto de las operaciones sobre el sistema. La recreación de los contratos de operaciones depende del desarrollo del modelo conceptual, diagrama de secuencias del sistema y de la identificación de las operaciones del sistema. Los artefactos realizados cuando estamos viendo el comportamiento del sistema consisten en descripción de ***que hace*** *el sistema, sin extendernos en* ***como lo hace*** *(vemos al sistema como una caja negra).*

Los diagramas de secuencias desarrollados en el capítulo anterior muestran los eventos del sistema que el o los actores generan, pero no elabora un detalle de las funcionalidades asociadas con las operaciones invocadas. No podremos encontrar en estos diagramas los detalles necesario para poder entender la respuesta del sistema (el comportamiento del sistema). Para ello, en este capítulo mostraremos los contratos de las operaciones identificadas en el modelo de secuencias de sistemas. Un contrato es un documento que describe que es lo que hace una operación para llegar a su cometido. El énfasis está puesto en que es lo hace más en como lo hace.

Hay dos tipos de contratos posibles: los que son usados para escribir un método de una clase de software y los usados para revisar las operaciones del sistema.
Estos últimos describen los cambios de estado de todo el sistema cuando una operación de sistema es usada. En este capítulo, realizaremos los contratos de sistemas de las operaciones descubiertas durante la realización del diagrama de secuencias de sistemas (ver Capítulo 5).

6.2- Los contratos y otros artefactos.

Mirando la descripción de los casos de uso, podemos deducir los eventos del sistema y construir los diagramas de secuencia de sistemas. Mediante estos últimos (diagramas de secuencias del sistema) identificamos las operaciones del sistema cuyo efecto es descripto mediante sus respectivos contratos.

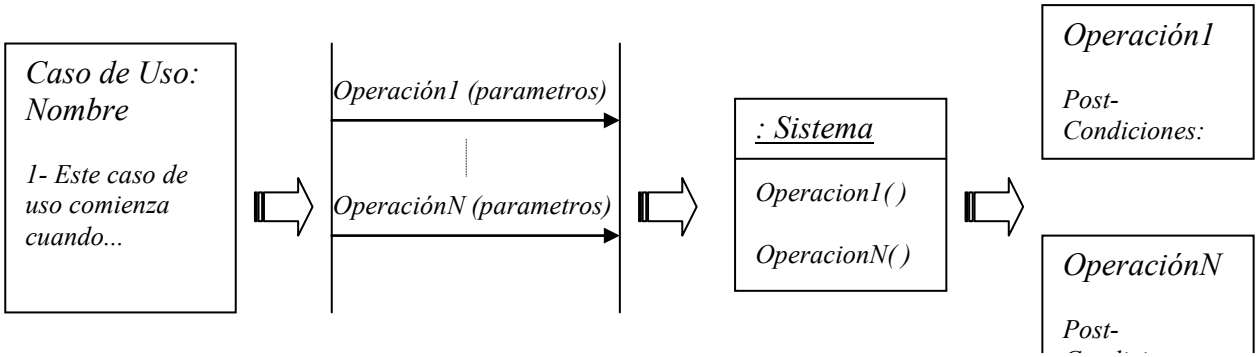


Figura 6.1
Relación entre contratos y otros artefactos

6.3- Secciones de un contrato.

A continuación mostraremos un esquema de un contrato y describiremos lo que encontraremos en cada sección

Nombre	*Nombre de la operación y parámetros*
Responsabilidades	*Descripción de las responsabilidades que la operación debe satisfacer*
Tipo	*Diremos el tipo de operación que vamos a realizar. En este momento dirá siempre Sistema porque realizaremos los contratos de las operaciones del sistema, pero también podría decir de Clase de software.*
Referencias cruzadas	*Referencias numéricas de funciones de sistema, nombres de casos de uso, etc.*
Notas	*Notas de diseño, algoritmos, etc.* *Ponemos algún algorítmico o alguna sentencia (se utiliza cuando por ejemplo se desea que una operación se realice en determinada manera)*
Excepciones	*Casos excepcionales. Se utiliza para discutir la reacción a situaciones excepcionales*
Salidas	*Salidas que no son interfase de usuario, tales como mensaje o registros que son enviados fuera del sistema*
Condiciones previas (pre-condiciones)	*Asunciones que hacemos acerca del estado del sistema antes de la ejecución de la operación*
Condiciones posteriores (post-condiciones)	*Enunciamos el estado del sistema luego de completada la operación.*

De todas las secciones descriptas anteriormente, hay muchas que en algunos casos son opcionales. Hay que tener en cuenta que la descripción de responsabilidades, las pre-condiciones y las post-condiciones son las secciones más importantes.

6.3.1- Respecto a las Post-condiciones.

Las post-condiciones muestran cómo el sistema ha cambiado una vez que una operación ha finalizado. ***No son acciones*** *a ser ejecutadas durante la operación, sino que son declaraciones del* ***estado*** *del sistema una vez que la operación ha concluído. Están orientadas a los estados, más que a las acciones.*

Las categorías de post-condiciones habituales son:

- *creación o borrado de instancias*
- *modificación de atributos*
- *formación o ruptura de asociaciones*

Están relacionados al modelo conceptual, dado que las instancias que se han creado o los atributos que se han fijado o modifican o las asociaciones que se han formado son los conceptos, atributos y asociaciones del modelo conceptual.

Al declarar el estado del sistema y no las acciones a seguir, las post-condiciones constituyen una herramienta excelente para mostrar que efectos que tiene una operación sin tener que describir como se hará la misma. Osea que lo que logramos, es diferir las decisiones de diseño y concentrarnos analíticamente en que hace la operación en vez de cómo se hace.

Cuando realicemos las post-condiciones de un contrato, tendremos que estudiar el modelo conceptual y establecer en orden:

1- *¿Qué instancias fueron creadas o borradas una vez que ocurrió la operación?*
2- *¿Qué atributos han sido modificados?*
3- *¿Qué asociaciones han sido creadas o rotas?*

6.4- Acerca de la organización del capítulo.

Realizaremos los contratos para las operaciones de sistemas encontrados en el capítulo 5, en el orden en que fueron mostrados anteriormente, a saber:

1º EntrarPeríodo
2º TerminarLiquidación
3º RealizarPago
4º EjecutarImputación

Cabe señalar que dividiremos las post-condiciones en instancias borradas y creadas, atributos modificados y asociaciones creadas o rotas.

Eventualmente, desarrollaremos alguna explicación adicional sobre el sistema para mejorar la comprensión del contrato.

Una vez que hayamos escrito todos los contratos, explicaremos cuales conceptos, atributos y asociaciones serán creados en el momento de inicio del equipo y se procederá a reveer el modelo conceptual para completarlo (en base a los descubrimientos realizados durante la creación de los contratos)

6.5- Realización de contrato para la operación de sistema "EntrarPeríodo".

Nombre:	*EntrarPeríodo* *(identificación del dominio: alfa-numérico* *período: numérico)*
Responsabilidades:	*Ingresar la liquidación de un período.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 1.1, 1.2, 1.3, 1.4, 1.6* *Casos de Uso: Liquidar Impuesto* *Pagar Impuesto*
Notas:	
Excepciones:	*Si la identificación del dominio es inválida indica que hay un error.*
Output:	
Pre-condiciones:	*La identificación del dominio es conocida por el sistema* *El período a ingresar existe para el vehículo.* *Las características del vehículo y los períodos emitidos para ese vehículo son conocidos por el sistema.*

Post-condiciones:

a) Instancias creadas o borradas una vez que se completo la operación del sistema:

- Si es una nueva Liquidación, una Liquidación fue creada (creación de instancia)

- Una LíneaPeríodoLiquidado fue creada (creación de instancia)

b) Modificación de los atributos

- Si fue una nueva Liquidación la Liquidación.ID_Vehículo fue fijado (modificación de atributo).

- Si fue una nueva Liquidación la Liquidación.VencimientoLiquidación fue fijado (modificación de atributo).

- *Si fue una nueva Liquidación, la liquidación.esConPago fue fijada.*
- *Si fue una nueva Liquidación, la Liquidación.FondoEducativo fue fijado (modificación de atributo).*
- *La Liquidación.TotalaAbonar fue modificado (modificación de atributo).*
- *La LíneaPeríodoLiquidado.PeríodoLiquidado fue fijado (modificación de atributo).*
- *La LíneaPeríodoLiquidado.VencimientoOriginal fue fijado (modificación de atributo).*
- *La LíneaPeríodoLiquidado.DeudaOriginal fue fijado (modificación de atributo).*
- *La LíneaPeríodoLiquidado.IndiceLiquidación fue fijado (modificación de atributo).*

c) Asociaciones formadas o rotas

- *Si fue una nueva Liquidación, la nueva Liquidación fue asociada a la terminal (relación formada).*
- *Si fue una nueva Liquidación, la nueva Liquidación fue asociada a un Vehículo, basado en la identificación del vehículo(relación formada).*
- *La Liquidación fue asociada a la nueva LíneaPeríodoLiquidado,*
- *La LíneaPeríodoLiquidado fue asociada al PeríodoVehículo, basado en el período y identificación de vehículo(relación formada).*

Aclaraciones:

Note que ha aparecido un nuevo atributo para el concepto liquidación que hemos llamado "esConPago", el cual estará en true si el contribuyente realiza el pago en el momento (en la oficina de rentas) y en false si el contribuyente solo pide la liquidación (en consecuencia el pago lo realizará en el banco). Este atributo será utilizado en el momento de la imputación de pagos.

Por otro lado, hemos eliminado el atributo DeudaActualizada del concepto PeríodoLiquidado debido que este concepto resulta deducible del índice de actualización y la deuda original del período a liquidar

6.6- Realización de contrato para la operación de sistema "TerminarLiquidación".

6.6.1- Acerca de las Barras de la Liquidación.

Antes de realizar este contrato creemos conveniente revisar un poco el diagrama conceptual en lo que se refiere a las barras. Una barra es un objeto que se imprimirá en el momento de realizar la liquidación, si es que el pago no se realiza en la oficina de operación sino a través del banco. La idea es que el banco lea con un lector óptico dicha barra y que puede calcular el monto que el contribuyente debe pagar como así también asentar los datos para posteriormente ser enviados para su imputación a la oficina de operación.

En una liquidación existen 2 barras:

- *Barra General: contiene los datos generales de toda la liquidación*

  El importe contenido en esta barra es el que corresponden pagar a la fecha de pago que contiene la barra. En caso de que el contribuyente se presente con posterioridad a la fecha de barra el banco procederá a calcular el importe actualizada para realizar el correcto cobro del impuesto (se calcula el interés desde la fecha de barra a la fecha de pago del impuesto).

- *Barra Períodos Reducida: contiene los datos (reducidos) de todas las líneas de la liquidación. Estos datos son puestos en la barra para que el banco pueda enviarlos al ente recaudador con el propósito de validación antes de la imputación.*

Los datos de la barra general serán guardados como atributos de la liquidación y los de la barra período reducida son datos encontrados en las LíneasPeríodoLiquidado. En el momento de terminar con la entrada de períodos, el generador de barras generará las barras (si es que el pago se realiza vía banco), para ello Liquidación deberá mandarle en un string la decodificación de la o las barras que desea que genere.

6.6.2- Contrato de TerminarLiquidación.

Nombre:	*TerminarLiquidación*
Responsabilidades:	*Registrar el fin de la liquidación, mostrar y asentar los datos de la liquidación.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 1.1, 1.2, 1.6, 1.8* *Casos de Uso: Liquidar Impuesto* *Pagar Impuesto*
Notas:	
Excepciones:	*Si no se marca al menos un período indica un error.*
Output:	
Pre-condiciones:	*La identificación del dominio es conocida por el sistema*

Los períodos a liquidar son conocidos por el sistema.
Las características del vehículo y los períodos emitidos para ese vehículo son conocidos por el sistema.
Se ha marcado al menos un período.

Post-Condiciones

a) *Creación de instancias*
- Si Liquidación.esConPago se encontraba en falso, las 2 barras de la liquidación son creadas.

b) *Modificación de los atributos*
- *Liquidación.esCompletada fue fijada en true (modificación de atributo) (nuevo atributo).*
- *Si Liquidación.esConPago se encontraba en falso, la Liquidación.FechadeBarra fue fijada.*
- *Si Liquidación.esConPago se encontraba en falso, la Liquidación.Importedebarra fue calculada.*
- *La Liquidación.RealizoPago es fijada en falso (nuevo atributo).*
- *La Liquidación.TotalAbonar es calculado.*

c) *Asociaciones formadas o rotas*
- *Si Liquidación.esConPago se encontraba en falso, la Liquidación fue asociada al Generador de Barra (relación formada).*
- *Si Liquidación.esConPago se encontraba en falso, Generador de Barra fue asociada a la o las barras (relación formada).*
- *La Liquidación fue asociada al Catálogo de Liquidación (relación formada)*

Aclaraciones:

Al pensar este contrato hemos descubierto algunas cosas nuevas del modelo conceptual:

- *El atributo Liquidación.esCompletada: indica que se ha terminado de marcar los períodos a liquidar*
- *El atributo Liquidación.RealizoPago: indica si ya se ha realizado un pago o no para una liquidación dada. En el momento que se ingrese el pago de una liquidación, (ya sea on-line o batch) este atributo tomará el valor de verdadero.*
- *Existe una correspondencia entre una liquidación dada y su Pago. Por un lado, no se pueden realizar 2 pagos con una misma liquidación, esto sería un error (para evitar este error es que utilizamos el atributo RealizarPago) y por el otro no se puede realizar un pago de algo que no se liquidó o que se liquidó pero ya tiene pago asociado.*
- *La asociación la barra y la liquidación no existe, ya que la barra es un objeto de impresión, que se genera justo antes de imprimir el recibo mediante el generador de barra.*
- *El Generador de barra esta asociado a la barra*
- *El Liquidación está asociada al Generador de Barra.*

6.7- Realización de contrato para la operación de sistema RealizarPagos.

Responsabilidades:	*Registrar en el sistema el pago realizado por un contribuyente, realizar el balance e imprimir el recibo.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 1.5, 1.6, 1.8, 1.9, 2.1, 2.2, 2.3, 2.4* *Casos de Uso: Pagar Impuesto*
Notas:	
Excepciones:	*Si no el importe es menor que el liquidado es un error.*
Output:	
Pre-condiciones:	*La identificación del dominio es conocida por el sistema* *El período a liquidar es conocido por el sistema* *Las características del vehículo y los períodos emitidos para ese vehículo son conocidos por el sistema.* *La liquidación asociado es conocida por el sistema.*

Post-Condiciones

a) *Instancias creadas o borradas una vez que se completo la operación del sistema:*
   - *Un nuevo Pago online fue creado (creación de instancia)*

b) *Modificación de los atributos*
   El PagoOnline.ImporteAbonado fue fijado (modificación de atributo)
   El PeríodoVehículo.ImportePagado fue modificado para cada período liquidado (modificación de atributos)
   El PeríodoVehículo.FechaPago fue fijado a la fecha corriente (modificación de atributo)
   La Liquidación.RealizarPago es fijado en verdadero

c) *Asociaciones formadas o rotas*
   - *El PagoOnline fue asociado a la Liquidación.*
   - *El PagoOnline fue asociado a una terminal*
   - *El PagoOnline fue asociado a un Vehículo basado en la identificación del dominio*

Aclaraciones:

- *En este contrato ha aparecido el atributo FechaPago para el concepto PeríodoVehículo este atributo indica la fecha en que se realizó.*
- *Hemos incorporado una asociación entre Terminal y PagoOnline*

6.8- Realización de contrato para la operación de sistema ImputarPagos.

Nombre:	*ImputarPagos(datos del pago)*
Responsabilidades:	*Registrar en el sistema el pago realizado por un contribuyente, realizar el balance e imprimir el recibo.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 1.5, 1.8, 2.5, 2.6*
Notas:	
Excepciones:	
Output:	
Pre-condiciones:	*La identificación del dominio es conocida por el sistema* *Los períodos a pagar fueron previamente liquidados.* *Las características del vehículo y los períodos emitidos para ese vehículo son conocidos por el sistema.* *Las liquidaciones asociadas son conocidas por el sistema.*

Post-condiciones:

a) *Instancias creadas o borradas una vez que se completo la operación del sistema:*
   - *Un nuevo PagosBatch fue creado (creación de instancia)*
   - *Nuevos PeríodosPagoBatch fueron creados (creación de instancias)*
   - *Una Validación fue creada*

b) *Modificación de los atributos*
   - *El PagoBatch.IdentificaciónDominio fue fijado (modificación de atributo)*
   - *El PagoBatch.ImporteCobrado fue fijado(modificación de atributo)*
   - *El PagoBatch.FechaPago fue fijado (modificación de atributo)*
   - *El Pagobatch.Fechabarra fue fijado (modificación de Atributo)*
   - *El Pagobatch.ImporteBarra fue fijado (modificación de Atributo)*
   - *Los PeríodoPagoBatch.PeríodoImputar fueron fijados para todos los PeríodosPagoBatch(modificación de atributo).*
   - *Los PeríodoPagoBatch.DeudaOriginal fueron fijados para todos los PeríodosPagoBatch(modificación de atributo).*
   - *La liquidación.RealizarPago fue fijada en true.*
   - *La Validación.EsCorrecta fue fijada.*
   - *Los PeríodoVehículo.ImportePagado asociados a todos los PeríodosPagoBatch fueron actualizados (modificación de atributo).*
   - *Los PeríodoVehículo.FechaPago fueron fijados para todos los PeríodosPagoBatch ingresados (modificación de atributo).*

c) *Asociaciones formadas o rotas*
   - *El PagoBatch fue asociado al CatálogodePago*
   - *Si fue una nueva ejecución, el CatálogodePago fue asociado a la OficinaOperación.*

- *El PagoBatch fue asociado a cada PeríodoPagoBatch*
- *El PagoBatch fue asociado a la Terminal*
- *El PagoBatch fue asociado a la Liquidación basado en la Identificación del Vehículo, los LíneaPeríodosLiquidados(periodos liquidados, deuda original liquidada) y PeríodosPagoBatch , FechaBarra e Importe de Barra General de la liquidación y la FechaBarra e ImporteBarra del PagoBatch*
- *El PeríodoPagoBatch fue asociado a la LíneaPeríodoLiquidado*
- *El PagoBatch fue asociado a la Validación*
- *Los PeríodosPagoBatch fue asociado a la Validación*
- *Los PeríodosLiquidados fue asociado a la Validación*
- *La Validación fue asociada la Liquidación*
- *La Validación fue asociada a la Terminal*
- *El Vehículo fue asociado al PagoBatch basado en la identificación del vehículo*
- *Cada PeríodoPagoBatch fue asociado al PeríodoVehículo basado en la identificación del vehículo y en el período*

6.9- Realización de contrato para el Start up.

Para realizar los contratos de los casos de uso elegidos para desarrollar en el primer ciclo de desarrollo, hemos supuesto que el momento del encendido del equipamiento, se realizarán una serie de tareas que estaran encerradas en el caso de uso Start up y en la operación de sistema que posee el mismo nombre. En esta sección realizaremos el contrato de esa operación de sistema.

Nombre:	*StartUp*
Responsabilidades:	*El administrador, al comienzo del día, prende la máquina y realiza las operaciones de inicialización necesarias para que se pueda comenzar a operar el sistema.*
Tipo:	*Sistema*

Pre-condiciones:
Post-condiciones:

a) Conceptos creados
- *Un nueva Oficina fue creada*
- *Nuevas Terminales fueron creadas*
- *Un Catálogo de vehículo fue creado*
- *Un Catálogo de Liquidación fue creado*
- *Un Generador de barras fue creado*

b) Asociaciones Creadas
- *Cada Terminal fue asociada a la Oficina*
- *La Oficina fue asociada al Catálogo del Vehículo*
- *La Oficina fue asociada al Catálogo de la Liquidación*
- *La Oficina fue asociada al Generador de barras*

6.10- Conceptos, atributos y asociaciones incorporados o borrados del modelo conceptual.

Mediante el estudio de contratos hemos encontrado modificaciones al modelo conceptual, así han aparecido nuevos atributos, conceptos y atributos, a la vez que hemos considerado borrar conceptos existentes.

A continuación se enunciarán tales modificaciones y se mostrará la figura 6.2 con la nueva versión del modelo conceptual.

6.10.1- Nuevos Atributos

Se han encontrado los siguientes atributos nuevos para el concepto liquidación:

- *EsConPago*
- *EsCompletado*
- *RealizarPago*
- *ImporteBarra*
- *FechaBarra*

Se han encontrado los siguientes atributos nuevos para el concepto PeríodoVehículo:

- *FechaPago*
- *ImportePagado*

6.10.2- Nuevas Asociaciones

- *Generador de Barra se asocia Barra*
- *Liquidación se asocia a Generador de Barra*
- *PagoOnline se asocia Terminal*

6.10.2- Asociaciones eliminadas del modelo conceptual

- *Oficina fue asociado a PagoOnline: esta asociación está implícita mediante otras 2 asociaciones:*
  - *Oficina está asociada a la Terminal*
  - *Terminal registra PagoOnline*
- *Oficina fue asociada a Validación: esta asociación está implícita mediante otras 2 asociaciones:*
  - *Oficina está asociada a la Terminal*
  - *Terminal se realizan Validación*
- *Liquidación se asocia a la Barra: constituye un error conceptual ya que la barra es un objeto de impresión. Lo que contiene la Liquidación son los datos de la barra (decodificación de la barra) y no la barra propiamente dicha.*

6.10.4- Diagrama del Modelo Conceptual

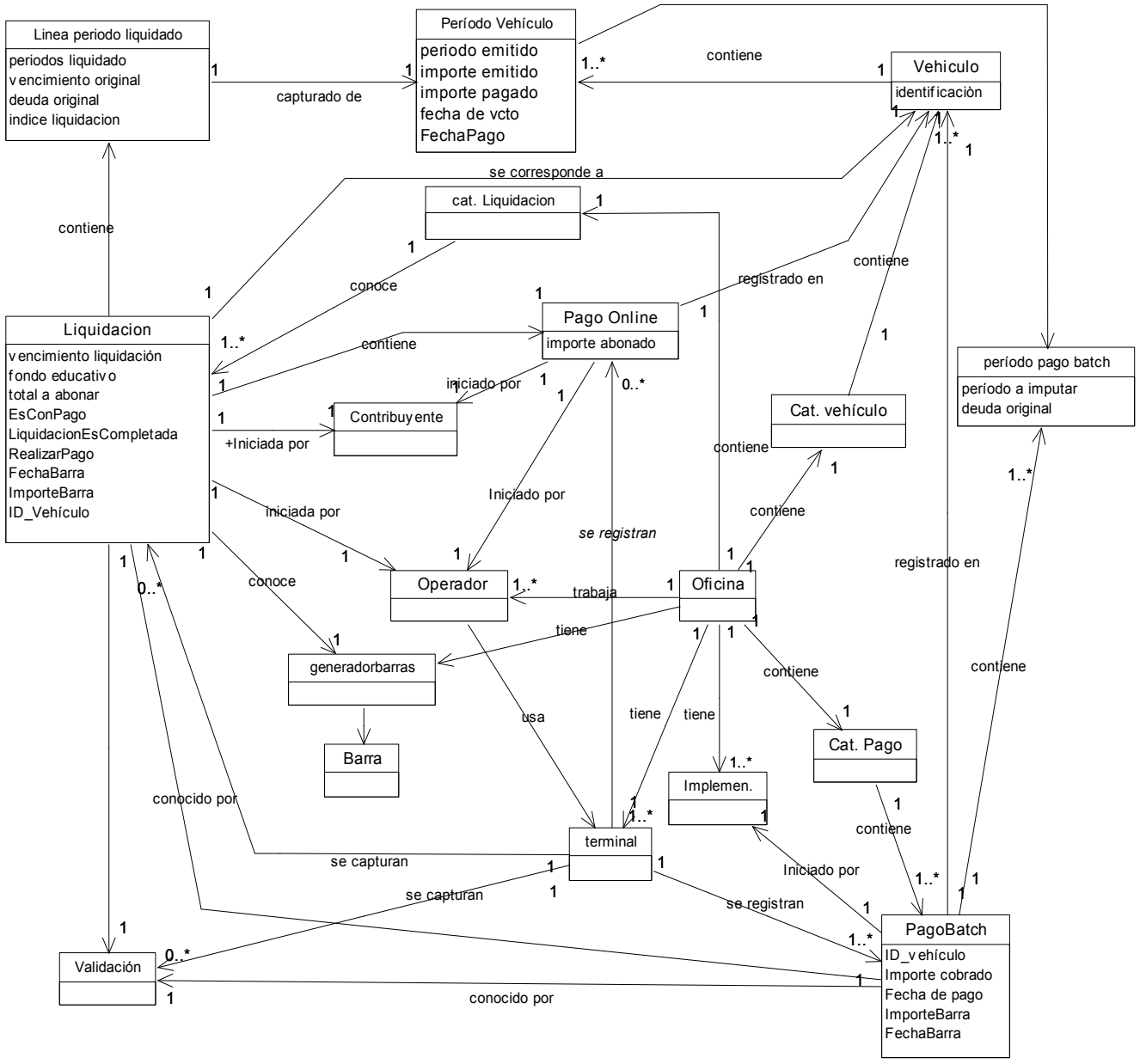


Figura 6.2

Capítulo 7 - Diagrama de colaboración

7.1- Introducción

En el capítulo anterior escribimos los contratos de las operaciones de sistema que estamos tratando en esta primera etapa del ciclo de desarrollo. Los mismos no muestran una solución de cómo los objetos de software hacen para completar las post-condiciones, sino los cambios que sufren los objetos del sistema en término de creación o borrado, atributos fijados, modificados o calculados y asociaciones formados o destruidas. Los diagramas de colaboración, en vez, ilustran como los objetos interactúan vía mensajes para completar la tarea.

Los diagramas de colaboración ocurren durante la fase de diseño del ciclo de desarrollo. Su creación depende de:

- *El modelo conceptual: desde allí saca las clases de software que se corresponden con los conceptos. Los objetos de estas clases participan de interacciones ilustradas en el diagrama de interacción.*
- *Los contratos de operaciones del sistema: en ellos se identifican las responsabilidades y post-condiciones que el diagrama de interacción debe satisfacer*

Un diagrama de interacción ilustra las interacciones de mensajes entre instancias (y clases) en el modelo de clase.

Existen 2 clases de diagramas de interacción:

1- *diagrama de colaboración*
2- *diagrama de secuencias*

Usaremos el diagrama de colaboración por su expresibilidad, su habilidad de cubrir más información contextual y su economía espacial relativa.

Dado que los diagrama de colaboración constituyen uno de los artefactos más importantes creados en el análisis y diseño orientado a objetos, luego el esfuerzo realizado en su generación insumirá un porcentaje importante del total del proyecto.

7.2- Los contratos y otros artefactos

- *Los diagrama de caso de uso sugieren los eventos del sistema los cuales son mostrados en el diagrama de secuencias*
- *El efecto de los eventos de sistema es descripto en los contratos de operación del sistema*
- *Los eventos del sistema representan mensajes que inicializan los diagramas de interacción los cuales ilustran como los objetos inteactúan para completar las tareas requeridas.*

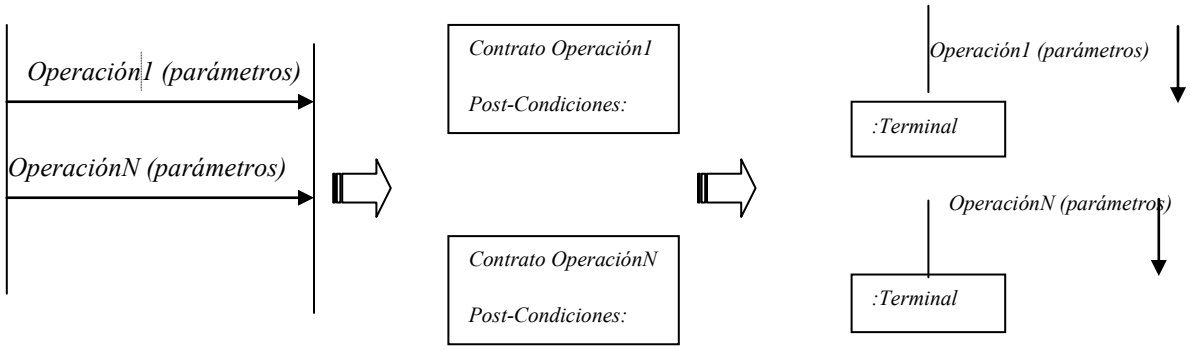


Diagrama de secuencia Del sistema ***Contratos*** ***Diagrama de colaboración***

Figura 7.1
Relación entre diagramas de colaboración y otros artefactos

7.3- Acerca de la organización del capítulo

En este capítulo realizaremos los diagramas de colaboración asociados a la liquidación e imputación de pagos del Impuesto a los Automotores.

En el anexo ***"Diagrama de colaboración"*** *podemos encontrar una breve explicación acerca de conceptos y sintaxis de dichos diagramas.*

Por otro lado en el anexo ***"Patrones Grasp"*** *[Larman97] se introducirán los conceptos necesarios para lograr tomar las decisiones de diseño necesarias en forma justificada (utilizando los patrones GRASP).*

7.4- Pasos a seguir para realizar el diagrama de colaboración

1- *Crearemos un diagrama de colaboración separado para cada operación de sistema en desarrollo en el ciclo de vida corriente (el diagrama comienza con el mensaje de la operación del sistema)*
2- *Si el diagrama se tornase complejo, lo dividiremos en diagramas más pequeños y simples*
3- *Usando las responsabilidades y post-condiciones y la descripción de los casos de uso, diseñaremos un sistema de objetos interactivos para completar las tareas. Para lograr este último objetivo nos valeremos de la aplicación de los patrones GRASP [Larman97] como una forma de justificar cada decisión (dichos patrones nos ayudarán visualizar los pro y contra de cada camino a tomar. Ver en Anexo "Patrones GRASP").*

7.5- Diagramas de colaboración de la iteración corriente

A continuación enunciaremos los diagramas de caso de uso y eventos del sistema asociados a dichos casos de uso:

- *Caso de uso: Liquidar Impuesto*
  - *EntrarPeríodo(identificaciónVehículo, Período)*
  - *TerminarLiquidación*

- *Caso de uso: Pagar Impuesto*
  - *EntrarPeríodo(identificaciónVehículo, Período)*
  - *TerminarLiquidación*
  - *RealizarPago()*

- *Caso de uso: Imputar Impuesto*
  - *EjecutarImputación(identificaciónVehículo, Período)*

Se realizarán los diagramas de colaboración de los siguientes eventos:

1- *EntrarPeríodo(identificaciónVehículo, Período)*
2- *TerminarLiquidación( )*
3- *RealizarPago()*
4- *EjecutarImputación(Datos del Pago)*

7.6- Diagramas de colaboración EntrarPeríodo

7.6.1- Mensaje de comienzo del diagrama

Será el evento del sistema EntrarPeríodo(IdentificaciónDominio, Período)

7.6.2- Elección de la clase controladora

Tenemos varias alternativas para la elección de la clase controladora de evento del sistema (o sea de la clase responsable de manejar el evento del sistema):

- *Utilizando el Controlador Facade podríamos elegir Clase **Terminal** ya que consideramos que ella representa a la interacción del usuario con el sistema.*
- *También utilizando el Controlador Facade podríamos elegir la Clase **Oficina** que representa a la organización*
- *Otra elección posible puede ser la clase **Operador** (utilizando en este caso el Controlador de Roles)*
- *Por último podríamos seleccionar un manejador del Caso de uso y así crear una clase llamada **ManejadorEntrarPeríodo***

*Elegiremos la primer alternativa, o sea que la clase controladora será **Terminal**.*

Los motivos de esta decisión son los siguientes:

La elección de clase manejadora de caso de uso fue lo primero que hemos descartado ya que el evento en estudio se encuentra en dos casos de uso diferentes: LiquidarImpuesto – PagarImpuesto

Debido a que existen pocas operaciones de sistema consideramos conveniente el uso del controlador facade ya que la clase controladora no tiene el riesgo de volverse cohesiva. (si esto ocurriese es conveniente utilizar otro controlador para así poder distribuir las responsabilidades entre las diferentes clases controladoras y no correr el riesgo de tener una clase con demasiadas responsabilidades).

Por el otro lado utilizando este controlador existen dos clases posibles: Terminal y Oficina. Elegiremos a la Terminal porque consideramos que es la clase que interactúa en forma directa con el actor.

7.6.3- Creación de una Nueva Liquidación

Cuando ocurre una nueva liquidación, ocurren las siguientes post-condiciones:

1- *Si fue una nueva Liquidación, una Liquidación fue creada*
2- *Si fue una nueva Liquidación, la nueva liquidación fue asociada a la terminal*
3- *Si fue una nueva Liquidación, la Liquidación.VencimientoLiquidación fue fijado*
4- *Si fue una nueva Liquidación, la Liquidación.esConPago fue fijada*
5- *Si fue una nueva Liquidación, la Liquidación.ID_Vehículo fue fijada*

Analicemos cada una de estas post-condiciones:

1- *Esta post-condición implica una responsabilidad de creación, y el patrón creador GRASP sugiere asignar la responsabilidad de creación a la clase que agrega, contiene o registra la clase a ser creada. Mirando el modelo conceptual observamos que en la terminal se registran las liquidaciones, luego podemos concluir que Terminal es la clase indicada para crear una nueva liquidación.*
   Por otro lado, en este momento, también debe crearse una colección vacía para registrar todos los períodos que serán ingresados. Nuevamente nos pondremos a pensar, quien sería la mejor clase para crear esta nueva colección. Dado que es mantenida por una instancia de la clase liquidación, es razonable pensar que dicha clase es la apropiada para la creación de la colección (por patrón creador).
2- *Teniendo la Terminal que crear una nueva Liquidación, fácilmente puede asociarse con ésta.*
3- *El patrón experto de los Patrones GRASP sugiere que se asigne la responsabilidad, al experto de la información: o sea a la clase que tiene la información necesaria para completar la responsabilidad. La clase Liquidación es la encargada de mantener sus atributos por lo tanto, por experto es razonable pensar que Terminal envíe un mensaje a Liquidación para que ella fije los valores de sus atributos. En*

consecuencia Terminal enviará el mensaje FijarVencimientoLiquidación(fecha) a la Liquidación.

4- *Por la misma razón del punto anterior podemos deducir que Terminal enviará el mensaje FijarEsConPago(Boolean) a la clase Liquidación*

5- *Por la misma razón se enviará el mensaje FijarID_vehículo(ID_Vehículo) a la clase Liquidación.*

El diagrama quedaría de la siguiente forma

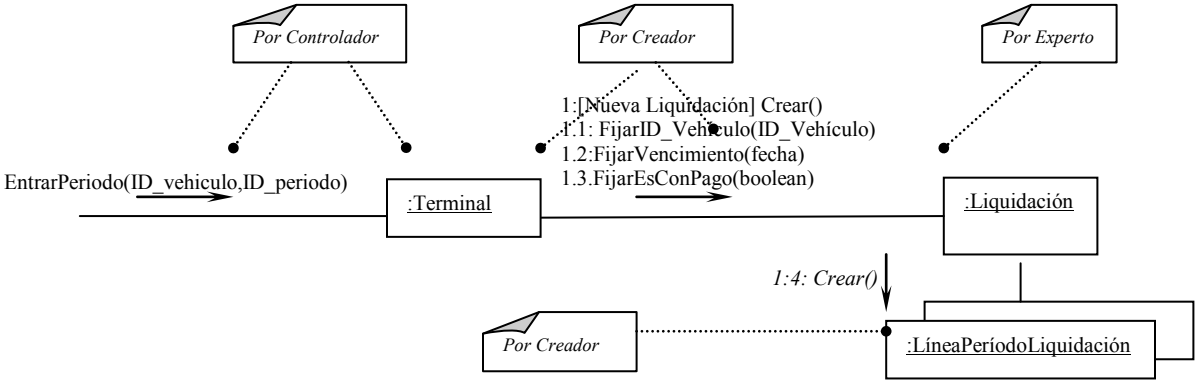


Figura 7.2
Diagrama de Colaboración de EntrarPeríodo - Versión 1

7.6.4- Obtención del Vehículo a Liquidar.

En el momento en que se realiza una nueva Liquidación es necesario obtener el objeto Vehículo a ser liquidado, ya que él posee los datos a cuales se hace referencia en la liquidación.

Mirando el modelo conceptual y las post-condiciones, vemos que existe una relación entre la Liquidación y el Vehículo cuyos períodos se están liquidando.

¿Quién sería responsable de obtener el Vehículo a liquidar?

Debido a que el CatálagoVehículo contiene todos los Vehículos, por experto deducimos que es una buena clase candidata.

¿Quién sería responsable de enviar el mensaje a la clase CatálogoVehículo para obtener el Vehículo a liquidar?

Podemos asumir que las instancias de CatálogoVehículo y Terminal fueron creadas durante el caso de uso Start up inicial y que existe una conexión entre la terminal y el CatálogoVehículo. Luego Terminal puede enviar el mensaje al CatálogoVehículo ya que ella tiene visibilidad sobre el catálogo.

Nuestro diagrama de colaboración quedaría de la siguiente manera:

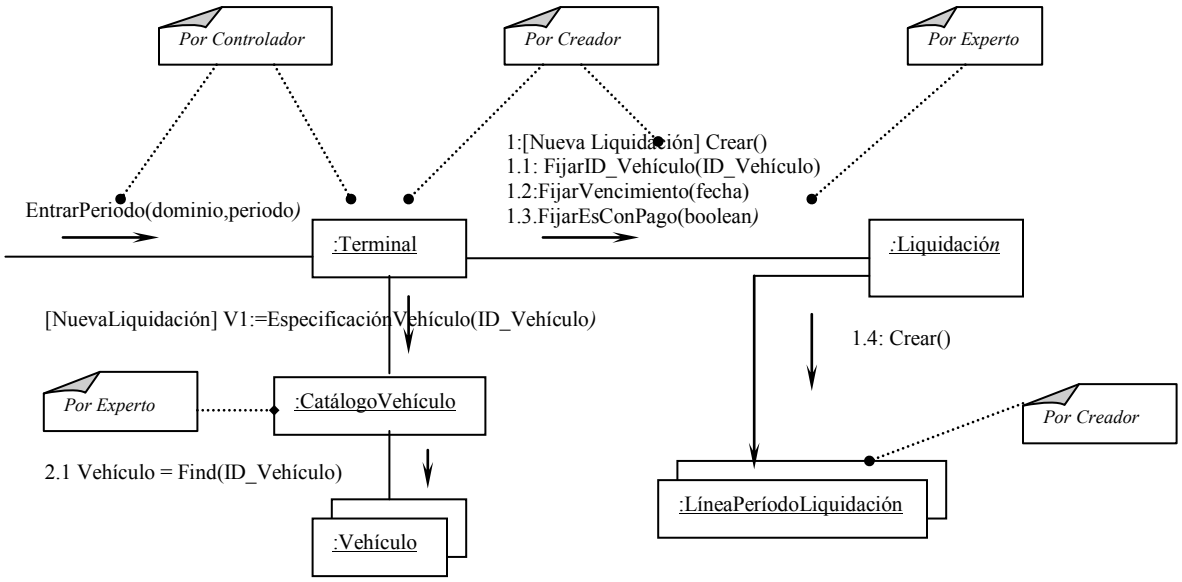


Figura 7.3
Diagrama de Colaboración de EntrarPeriodo - Versión 2

7.6.5- Creación de una Nueva LíneaPeríodoLiquidado

Cuando se crea una nueva instancia de LíneaPeríodoLiquidado, ocurren las siguientes post-condiciones:

1- *Si LíneaPeríodoLiquidado fue creada*
2- *La LíneaPeríodoLiquidado fue asociada a la liquidación fue fijado*
3- *La LíneaPeríodoLiquidado.PeríodoLiquidado fue fijado*
4- *La LíneaPeríodoLiquidado.VencimientoOriginal fue fijado*

La líneaPeríodoLiquidado necesita para su creación los atributos del período ingresado, los cuales se encuentran en una instancia de LíneaPeríodoVehículo que está dentro de una colección mantenida por la instancia Vehículo. LíneaPeríodoVehículo tendrá la responsabilidad de dar la información (por experto). Dichos atributos serán enviados por parámetro de entrada en el mensaje de creación. Liquidación será responsable de obtener esos datos solicitándoselos a la instancia de Vehículo vía mensaje (esto es factible gracias a que existe una conexión entre la instancia de Liquidación y de Vehículo, entonces instancia de Liquidación tiene visibilidad sobre la instancia de Vehículo)

¿Quién tendrá la responsabilidad de solicitarle a la colección la instancia de la LíneaPeríodoVehículo que se está liquidando?

Dado que la instancia de Vehículo mantiene la colección de LíneaPeríodoVehículo, por experto concluimos que es la clase candidata de tener tal responsabilidad.

¿Quién tendrá la responsabilidad de solicitarle a la instancia de LíneaPeríodoVehículo los valores de sus atributos fecha de vencimiento y deuda original respectivamente?

Dado que la instancia de Vehículo liquidado contiene las instancias de LíneaPeríodoVehículo, es lógico pensar por experto que Vehículo es candidato a tener tal responsabilidad.

Mirando el modelo conceptual, vemos que la Liquidación contiene 1 o varias LineasPeriodosLiquidado; luego, por creador, es la clase candidata para tener la responsabilidad de crear una nueva instancia de LíneaPeríodoLiquidado.

Dado que la Liquidación contiene una colección de objetos LíneaPeríodoLiquidado, los cuales ella mantiene; luego (por patrón Creador) la clase liquidación es clase candidata de tener la responsabilidad de enviar el mensaje a la colección de objetos para que se sume una nueva LíneaPeríodoLiquidado.

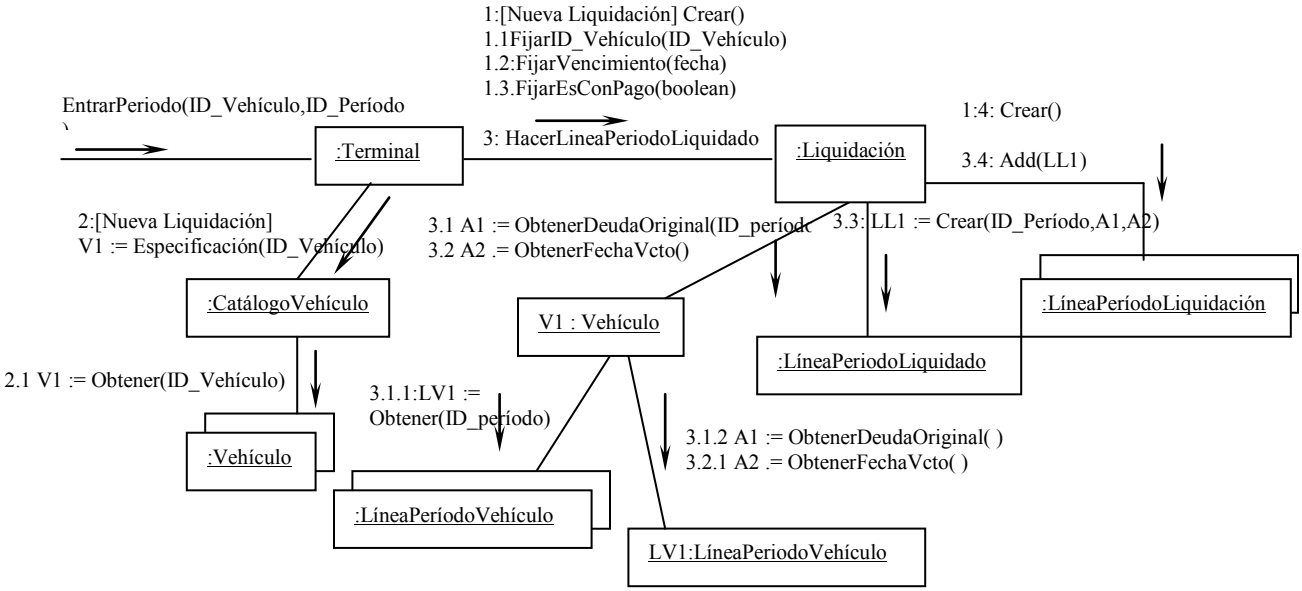


Figura 7.4
Diagrama de Colaboración final de EntrarPeríodo

7.7- Diagramas de colaboración TerminarLiquidación

7.7.1- Mensaje de comienzo del diagrama

Será el evento del sistema TerminarLiquidación()

7.7.2- Elección de la clase controladora

Por lo expuesto en el punto 7.5.1.2 eligiremos la clase ***Terminal.***

7.7.3- Diagrama de colaboración

Cuando Terminal recibe el mensaje de TerminarLiquidación, éste tiene la responsabilidad de informarle a Liquidación el fin del ingreso de los períodos. La Liquidación, por experto, es responsable de mantener sus atributos, luego es responsable de:

- *Fijar su atributo EsCompletado en true*
- *Fijar su atributo RealizarPago en false*
- *Fijar su atributo TotalAbonar (previo el cálculo del valor)*
- *Fijar su atributo ImporteBarra (si es que el pago se realizará en el banco y no en la oficina de operación) para ello se deberá calcular el valor*
- *Fijar su atributo FechaBarra (si es que el pago no se realizará en la oficina de operación, o sea que el atributo RealizarPago se encuentra en false).*

Por otro lado, si el pago se realiza en el banco, es necesario la generación de la barra, tarea que realizará (por patrón creador) el Generador de Barras. La Barra constituye un objeto que será impreso en la Liquidación y el cual será leído por el Banco, (con un dispositivo lector de barras) con la meta de obtener los datos de la Liquidación que se está pagando. El Banco calculará el importe a abonar utilizando la información leída y almacenará todos los datos, para su posterior envío al Ente Recaudador (para que éste proceda a la imputación de los mismos). El Generador de barras transforma una cadena de caracteres en una Barra. Luego nuestra pregunta es quien le proveerá la cadena (que llamaremos la decodificación de la barra). Dado que Liquidación posee todos los datos para el armado de la decodificación de la barra, por experto, es la clase indicada para tener la responsabilidad de crear la cadena y enviarla al Generador de Barra, quien procederá a transformar la cadena en una Barra.

Por último, la Liquidación debe ser guardada, para la posterior comparación en el momento de imputación. Dado que el catálogo de Liquidación contiene una colección de liquidación es el lugar adecuado para almacenar la liquidación(por experto).

¿Quién le solicitará al catálogo que sume una nueva instancia?

Dado que el catálogo es mantenido por la Oficina, será esta última clase la responsable de tal tarea(por experto).

Por último es factible pensar que en el momento del Startup del equipo se realice una conexión entre Terminal y Oficina, luego dado que Terminal tiene visibilidad sobre oficina es razonable que este le envíe un mensaje con la instancia de Liquidación para que Oficina le solicite al catálogo que lo sume a su colección.

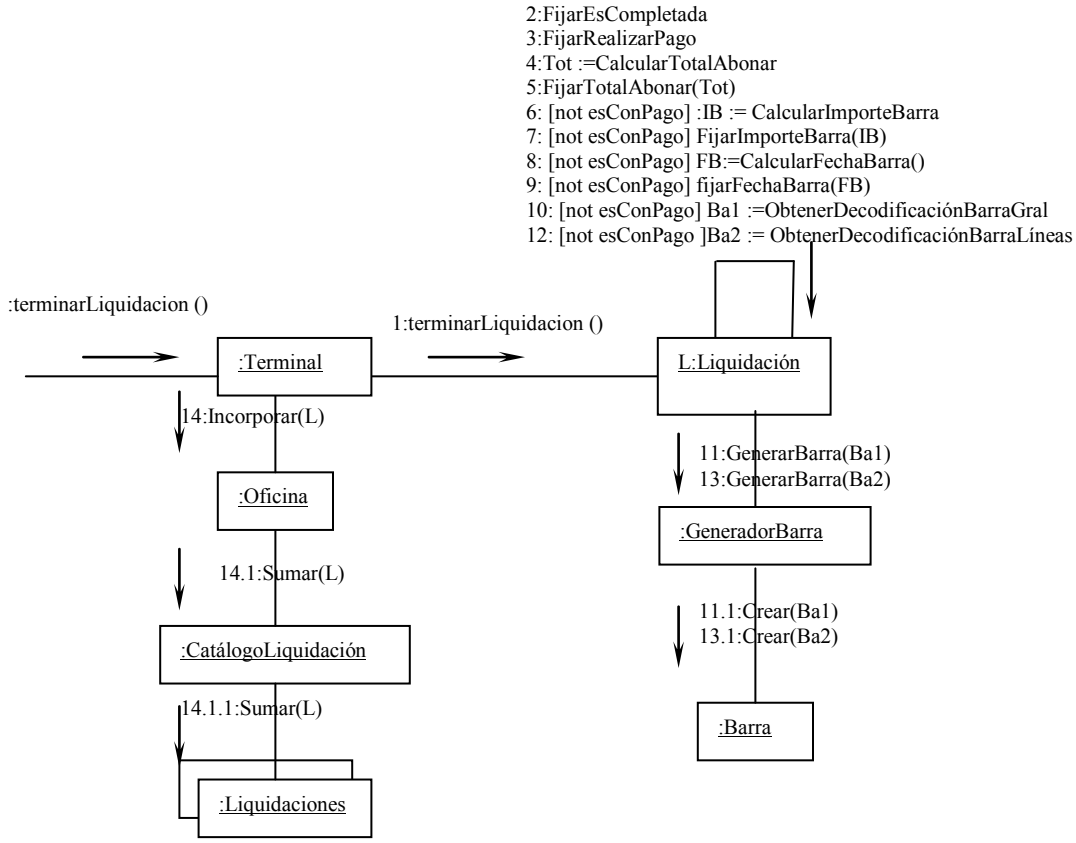


Figura 7.5
Diagrama de Colaboración final de TerminarLiquidación

7.8- Diagramas de colaboración RealizarPago

7.8.1- Mensaje de comienzo del diagrama

Será el evento del sistema RealizarPago()

7.8.2- Elección de la clase controladora

Por lo expuesto en el punto 7.5.1.2 elegiremos la clase ***Terminal.***

7.8.3- Envío el mensaje de realización de pago

Dado que los pagos serán asentados en Vehículo, él es una clase indicada para recibir tal mensaje. Por el otro lado, Liquidación tiene visibilidad sobre Vehículo, ya existe una conexión entre Liquidación y Vehículo (de hecho durante el ingreso de períodos Liquidación envío el mensaje a Vehículo solicitándole datos).

Tenemos dos clases candidatas:

- Vehículo
- Liquidación

Si seleccionamos Liquidación para realizar el trabajo, alivianamos a la clase Terminal.

Si elegimos Vehículo se requerirá que no solo la Terminal posea visibilidad de Vehículo, sino que Vehículo tendrá que ver a la Liquidación (hasta ahora era Liquidación la que veía al Vehículo y no viceversa)

Por lo expuesto llegamos a la conclusión de que Liquidación es la clase indicada para recibir el mensaje de realizarPago.

Nuestro diagrama de colaboración queda por ahora así:

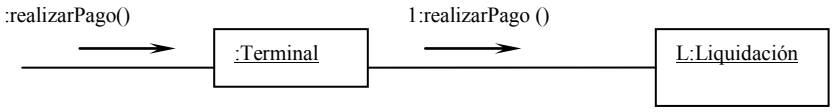


Figura 7.6
Diagrama de Colaboración de RealizarPago - Versión1

7.8.4- Actualización del pagado de cada línea liquidada

Consideremos las siguientes post-condiciones:

- *El LíneaPeríodoVehículo.importePagado fue modificado para cada período liquidado.*
- *- El LíneaPeríodoVehículo.fechaPago fue modificado para cada período liquidado.*

Una vez que se termina de marcar los períodos a liquidar, si se desea realizar un pago en el momento, será necesario de que los pagos queden asentados.

LíneaPeríodoVehículo posee todos los datos relacionados a un período dado. Entre estos datos se encuentran los relacionados a el pago (la Fecha de Pago y el Importe Pagado).

La pregunta es: ¿Quién será responsable de enviar el mensaje a LíneaPeríodoVehículo para que ella modifique sus atributos?

Dado que Vehículo contiene 1 o varias instancias de LíneaPeríodoVehículo, entonces por experto es la clase indicada para realizar tal tarea.

Por el otro lado, dado que Vehículo posee la colección de instancias de LíneaPeríodoVehículo, él es responsable de mantener dicha colección. Por experto, enviará un mensaje solicitándole que le devuelva la instancia de la LíneaPeríodoVehículo del cual queremos modificar los atributos.

Para saber la instancia de periodo que deseamos modificar, el importe y la fecha de pago a modificar, es necesario que Vehículo conozca el identificador de período, el importe pagado y la fecha de pago.

Ahora bien, ¿Quién le proveerá dicha información?

Dado que existe una conexión entre Vehículo y Liquidación, por visibilidad es factible que Liquidación le envíe un mensaje a Vehículo para que se actualice los datos y que dicho mensaje posea como parámetros los datos necesarios para la actualización.

¿Cómo obtendrá Liquidación los datos de la identificación de período, el importe a pagar y de la fecha de pago?

- *La fecha de pago será la fecha de liquidación (atributo de la Liquidación), que en este caso será la fecha corriente.*
- *El importe a imputar y el período al cual se le imputará, se sacará de cada una de las líneas de períodos liquidados.*

Para ello será necesario que para cada uno de los elementos de la colección de LíneaPeríodoLiquidado:

- *Se recupere la instancia de LíneaPeríodoLiquidado de la colección*
- *Se solicite a dicha instancia que devuelva el valor del Importe Original y el Identificador de período.*

Nuestro diagrama de colaboración quedaría así:

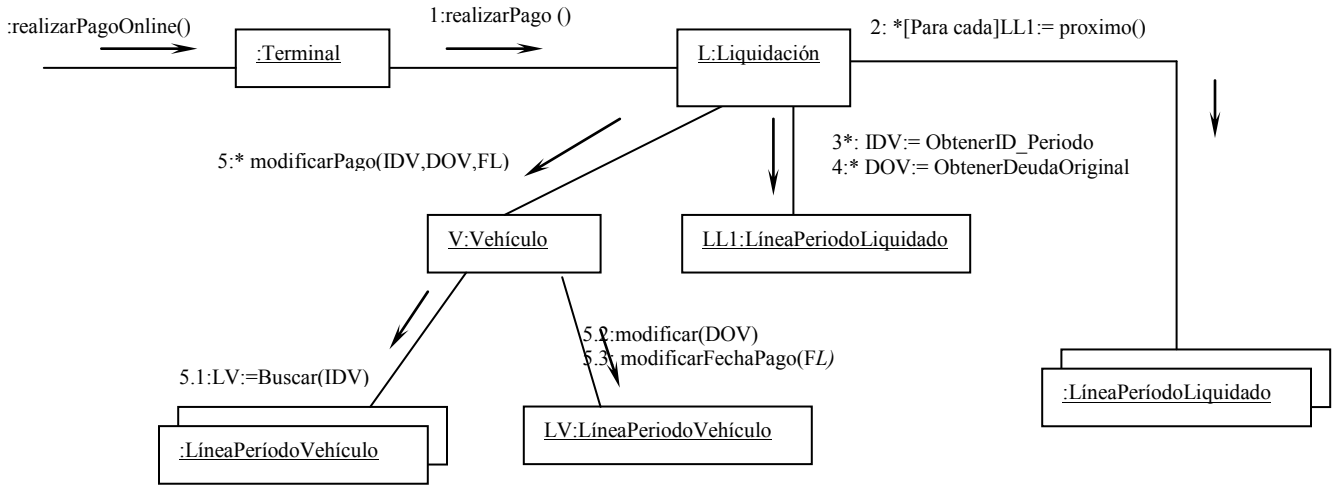


Figura 7.7
Diagrama de Colaboración final de RealizarPago

7.9- Diagramas de colaboración ImputarPago

Mirando el contrato de esta operación de sistema, nos encontramos con que existen 2 procesos que se realizan en la imputación de un pago: la carga del pago en el catálogo de PagoBatch y la Imputación propiamente dicha. Estos procesos se encuentran englobados en un único caso de uso, lo cual no nos satisface. Además nos damos cuenta que el concepto Validación del modelo conceptual posee funcionalidades que son más apropiadas que las realice otras clases. Hemos decidido reveer todo lo relacionado a esta operación de sistema, comenzando con el caso de uso asociado (ImputaciónPago). Esta revisión se hará en el capítulo siguiente y es allí donde estudiaremos este diagrama de colaboración.

7.10- Otros diagramas de colaboración

7.10.1- Obtención del importe de la barra general y del total de la liquidación

Lo primero que pensamos es en realizar un diagrama de colaboración para reflejar el importe de la barra general y otro distinto para el importe total de la liquidación. Pero examinemos un poco:

¿Cómo se obtiene el importe de la barra general?

El importe de la barra general se forma sumando los importes actualizados de cada período, a la fecha de barra general.

Osea que para cada línea liquidada se obtendrá un subtotal de la siguiente manera:

- *Se obtendrá la línea que se liquidó*
- *Se obtendrá el importe actualizado (este será el importe original + el interés desde la fecha de vencimiento del período a la fecha de la barra)*

¿Cómo se obtiene el importe total de la liquidación?

El importe total de la liquidación se forma sumando los importes actualizados de cada período liquidado a la fecha de vencimiento de la liquidación.

Osea que para cada línea liquidada se obtendrá un subtotal de la siguiente manera:

- *Se obtendrá la línea que se liquidó*
- *Se obtendrá el importe actualizado (este será el importe original + el interés desde la fecha de vencimiento del período a la fecha de vencimiento de la liquidación)*

Notemos que en realidad en ambos casos, estamos hablando de una operación que devuelve el total de una liquidación. La diferencia radica, en el interés que se le otorga a uno o a otro total (uno es hasta la fecha de la barra y el otro es hasta la fecha de liquidación). Luego hemos decido realizar una única operación de sistema, a la cual entrará como parámetro una fecha de actualización(fecha hasta donde se calcula el interés).

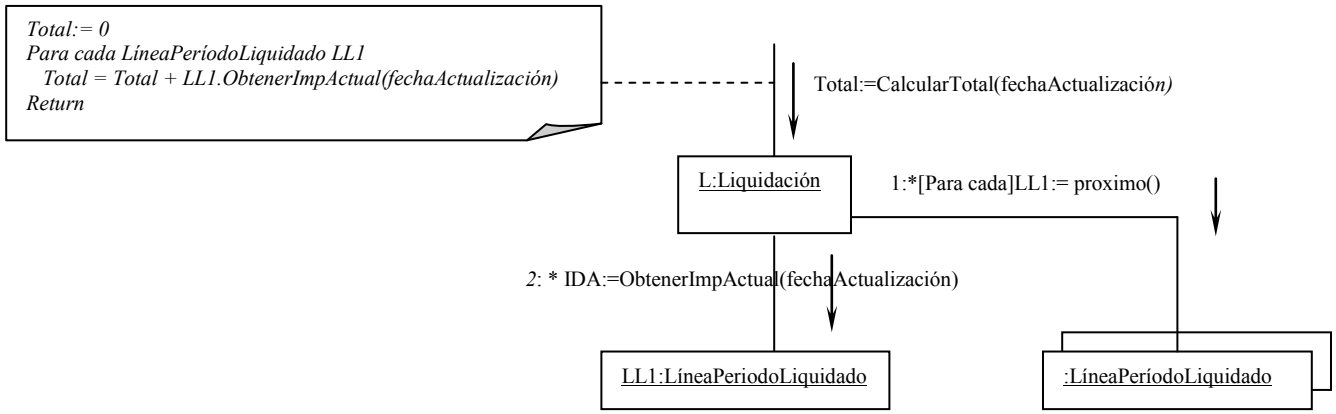


Figura 7.8
Diagrama de Colaboración de Calcular Total

En este diagrama se ve que liquidación es la responsable de conocer su total (por experto). Cuando una instancia de liquidación reciba el mensaje CalcularTotal (con la fecha de actualización del total como parámetro), dicha instancia le solicitará a cada una de las LíneaPeríodoLiquidado que le devuelva el importe original más el interés a la fecha de actualización (ya que por experto LíneaPeríodoLiquidado es la clase adecuada para obtener este importe), enviándole el mensaje ObtenerImpActual y los ira sumando uno a uno a su total.

7.10.2- Obtención de la decodificación de la Barra de Períodos Reducida

La decodificación de la barra constituye una secuencia de caracteres, que serán enviados al Generador de barras para que los transforme en una barra.

Como ya dijimos, en una Liquidación existen 2 barras:

- *Barra General: contiene los datos generales de la liquidación y la decodificación de la barra se obtendrá mediante la concatenación de los datos que son atributos de la liquidación.*
- *Barra Período Reducidos: contiene los datos (reducidos) de todas las líneas liquidadas. Estos datos son puestos en la barra para que el banco pueda enviarlos al ente recaudador con el propósito de validación antes de la imputación.*

En esta sección desarrollaremos el diagrama de colaboración de la operación ObtenerDecodificaciónBarraLíneas. Esta operación implica el acceso a cada período liquidado para poder generar la cadena de entrada al Generador de Barra. No la quisimos incorporar al diagrama de colaboración general para no complejizar dicho diagrama en demasía.

Como vimos en el diagrama de colaboración de TerminarLiquidación, Liquidación posee la responsabilidad de obtener la decodificación de la barra de líneas (la cadena de caracteres que contiene los datos reducidos de las líneas de períodos liquidados). Dicha decodificación se obtiene mediante la concatenación de los períodos y las deudas originales liquidadas. Mirando el modelo conceptual vemos que los datos necesarios se encuentran el cada una de las LíneasPeríodoLiquidado. Las LíneasPeríodoLiquidado están contenidas en la colección de LíneasPeríodosLiquidados, mantenida por la Liquidación. Entonces, por experto, Liquidación es responsable de obtener cada LíneaPeríodoLiquidado y de solicitarle los datos necesarios para concatenar (el período y la deuda original liquidado). Puesto que, LíneaPeríodoLiquidado posee como atributos el período y la deuda original, por patrón de experto, es la clase indicada para devolver dicha información.

El diagrama quedará de la siguiente forma:

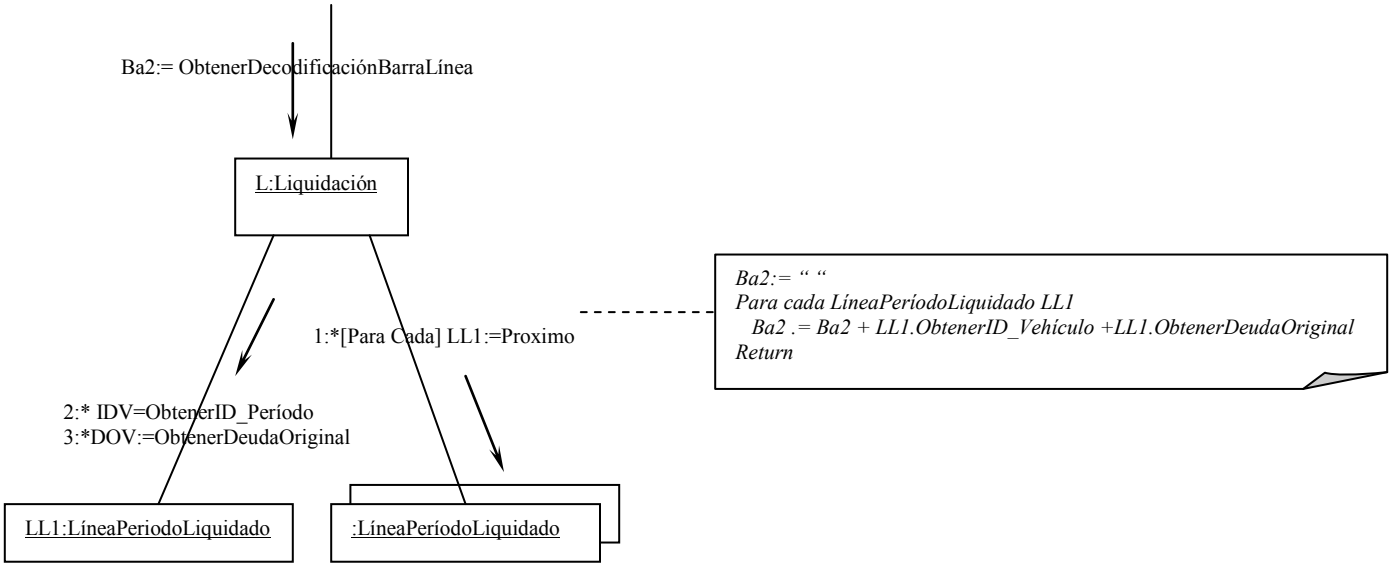


Figura 7.9
Diagrama de Colaboración de ObtenerDecodificaciónBarraLínea

7.11- Revisión del diagrama conceptual y de contratos

Antes de concluir con este capítulo queremos hacer una breve revisión de los contratos de las operaciones de sistema a las cuales les realizamos en diagrama de colaboración y en el modelo conceptual del sistema, para incorporar los conocimientos adquiridos durante la realización de los diagramas de colaboración.

7.11.1- Revisión del modelo Conceptual

En este capítulo, nos dimos cuenta que había asociaciones que en realidad no eran usadas. Por ejemplo, la forma de acceso a los períodos de un vehículo es a través del Vehículo, así si LineaPeríodoLiquidado necesitara datos que posee el PeríodoVehículo, no accederá directamente al período del vehículo sino que deberá solicitarle (de alguna manera) los datos al Vehículo, para que él a su vez se los obtenga del PeríodoVehículo.

De esta manera desaparecen del modelo conceptual las asociaciones entre las siguientes clases:

- *LíneaPeríodoLiquidado y PeríodoVehículo*
- *PeríodoVehículo y PeríodoPagoBatch*

Por otro lado, nos dimos cuenta que Validación no tiene funcionalidad ya que todas las operaciones que ella realiza, en realidad es más apropiado que la realice otra clase (este concepto nos servirá en el próximo capítulo). Por ejemplo: una de las cosas que hace Validación es calcular el importe que debió ser cobrado por el banco y compararlo con el importe que el banco reporta como que cobró efectivamente. Para calcular este supuesto cobrado utiliza los atributos de PagoBatch: fecha de barra, importe de barra y fecha de pago, y luego compara el resultado obtenido con otro atributo de PagoBatch (importe cobrado). Note que todos los datos del calculo están contenidos en PagoBatch, luego el patrón de experto sugiere que PagoBatch es la clase adecuada para realizar esta tarea, y no la Validación a la que, eventualmente, le tendríamos que pasar por parámetro todos los datos para que realice el cálculo. Luego Validación no es en realidad un concepto y decidimos eliminarla del modelo conceptual.

Nos dimos cuenta además que PagoOnline, tampoco tiene ninguna funcionalidad más que guardar el importe abonado por el contribuyente, dato que es deducible por otra vía. Esto nos motivó a que lo eliminaramos del modelo conceptual.

Hemos eliminado el índice de actualización como atributo de LíneaPeriodoLiquidado, ya que es un dato deducible de otros e incorporamos el atributo fecha que se realizó la liquidación en la Liquidación.

El modelo conceptual nos queda como muestra la figura 7.10 (en la hoja final de este capítulo).

7.11.2- Revisión de los Contratos

Teniendo en mente las modificaciones incorporadas al modelo conceptual, mostraremos los cambios de los contratos. Para ello, remarcaremos con letra negrita las post-condiciones que se agregan y tacharemos aquellas que existían y ahora no existen más.

7.11.2.1- Revisión del Contrato de EntrarPeríodo

Nombre:	*EntrarPeríodo (identificación del dominio: alfa-numérico período: numérico).*
Responsabilidades:	*Ingresar la liquidación de un período y sumar el importe liquidado al total de la liquidación.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 1.1, 1.2, 1.3, 1.4, 1.6* *Casos de Uso: Liquidar Impuesto* *Pagar Impuesto*
Excepciones:	*Si la identificación del dominio es inválida indica que hay un error.*

Pre-condiciones: *La identificación del dominio es conocida por el sistema*

El período a ingresar es conocido por el sistema
Las características del vehículo y los períodos emitidos para ese vehículo son conocidos por el sistema.

Post-condiciones:

- *Si fue una nueva Liquidación, una Liquidación fue creada.*
- *Si fue una nueva Liquidación, la nueva Liquidación fue asociada a la terminal.*
- *Si fue una nueva Liquidación, la nueva Liquidación fue asociada a un Vehículo, basado en la identificación del vehículo.*
- *Si fue una nueva Liquidación la Liquidación.ID_Vehículo fue fijado.*
- *Si fue una nueva Liquidación la Liquidación.esConPago fue fijado.*
- ***Si fue una nueva Liquidación la Liquidación.VencimientoLiquidación fue fijado**.*
- *Si fue una nueva Liquidación, la Liquidación.FechaLiquidación fue fijado.*
- ~~*Si fue una nueva Liquidación, la Liquidación.TotalaAbonar fue modificado*~~
- *Si fue una nueva Liquidación, la Liquidación.FondoEducativo fue fijado.*
- *Una LíneaPeríodoLiquidado fue creada.*
- ~~*La LíneaPeríodoLiquidado fue asociada al PeríodoVehículo basado en la identificación del vehículo y del período*~~
- *La Liquidación fue asociada a la nueva LíneaPeríodoLiquidado.*
- *La LíneaPeríodoLiquidado.PeríodoLiquidado fue fijado.*
- *La LíneaPeríodoLiquidado.VencimientoOriginal fue fijado.*
- *La LíneaPeríodoLiquidado.DeudaOriginal fue fijado.*
- ~~*La LíneaPeríodoLiquidado.IndiceActualización fue fijado.*~~

7.11.2.2- Revisión del Contrato de TerminarLiquidación

Nombre:	*TerminarLiquidación*
Responsabilidades:	*Registrar el fin de la liquidación, mostrar y asentar los datos de la liquidación.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 1.1, 1.2, 1.6, 1.8* *Casos de Uso: Liquidar Impuesto* *Pagar Impuesto*
Excepciones:	*Si no se marca al menos un período indica un error.*

Pre-condiciones: *La identificación del dominio es conocido por el sistema*
Los períodos a liquidar son conocidos por el sistema
Las características del vehículo y los períodos emitidos para ese vehículo son conocidos por el sistema.

Post-condiciones:

- *Liquidación.esCompletada fue fijada en true (modificación de atributo) (nuevo atributo)*
- ***La Liquidacion.TotalAbonar fue calculado***

- *Si Liquidación.esConPago se encontraba en falso, la Liquidación.FechadeBarra fue fijada.*
- *Si Liquidación.esConPago se encontraba en falso, la Liquidación.Importedebarra fue calculada.*
- *Liquidación.RealizoPago es fijada en falso (nuevo atributo).*
- *Si Liquidación.esConPago se encontraba en falso, la Liquidación fue asociada al Generador de Barra (relación formada)*
- *Si Liquidación.esConPago se encontraba en falso, las 2 barras fueron creadas*
- *Si Liquidación.esConPago se encontraba en falso, Generador de Barra fue asociada a la o las barras (relación formada)*
- *La Liquidación fue asociada al catálogo de liquidación (relación formada)*

Aclaración: *decidimos calcular en total a abonar una vez que se han ingresado todos los datos.*

7.11.2.3- Revisión del Contrato de RealizarPago

Nombre:	*RealizarPagos*
Responsabilidades:	*Registrar en el sistema el pago realizado por un contribuyente, realizar el balance e imprimir el recibo.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 1.5, 1.6, 1.8, 1.9, 2.1 a 2.4* *Casos de Uso: Pagar Impuesto*
Excepciones:	*Si no el importe es menor que el liquidado es un error.*
Pre-condiciones:	*La identificación del dominio es conocido por el sistema.* *El período a pagar existe para el vehículo ingresado.* *Las características del vehículo y los períodos emitidos son conocidos por el sistema.* *Las liquidaciones realizadas son conocidas por el sistema.*

Post-condiciones:

- ~~*Un nuevo Pago online fue creado (creación de instancia)*~~
- ~~*El PagoOnline fue asociado a una terminal*~~
- ~~*El PagoOnline fue asociado a la Liquidación*~~
- ~~*El PagoOnline.ImporteAbonado fue fijado (modificación de atributo)*~~
- ~~*El PagoOnline fue asociado a un Vehículo basado en la identificación del dominio*~~
- *El PeríodoVehículo.ImportePagado fue modificado para cada período liquidado (modificación de atributos)*
- *El PeríodoVehículo.FechaPago fue fijado a la fecha corriente (modificación de atributo)*
- *La Liquidación.RealizarPago es fijado en verdadero*

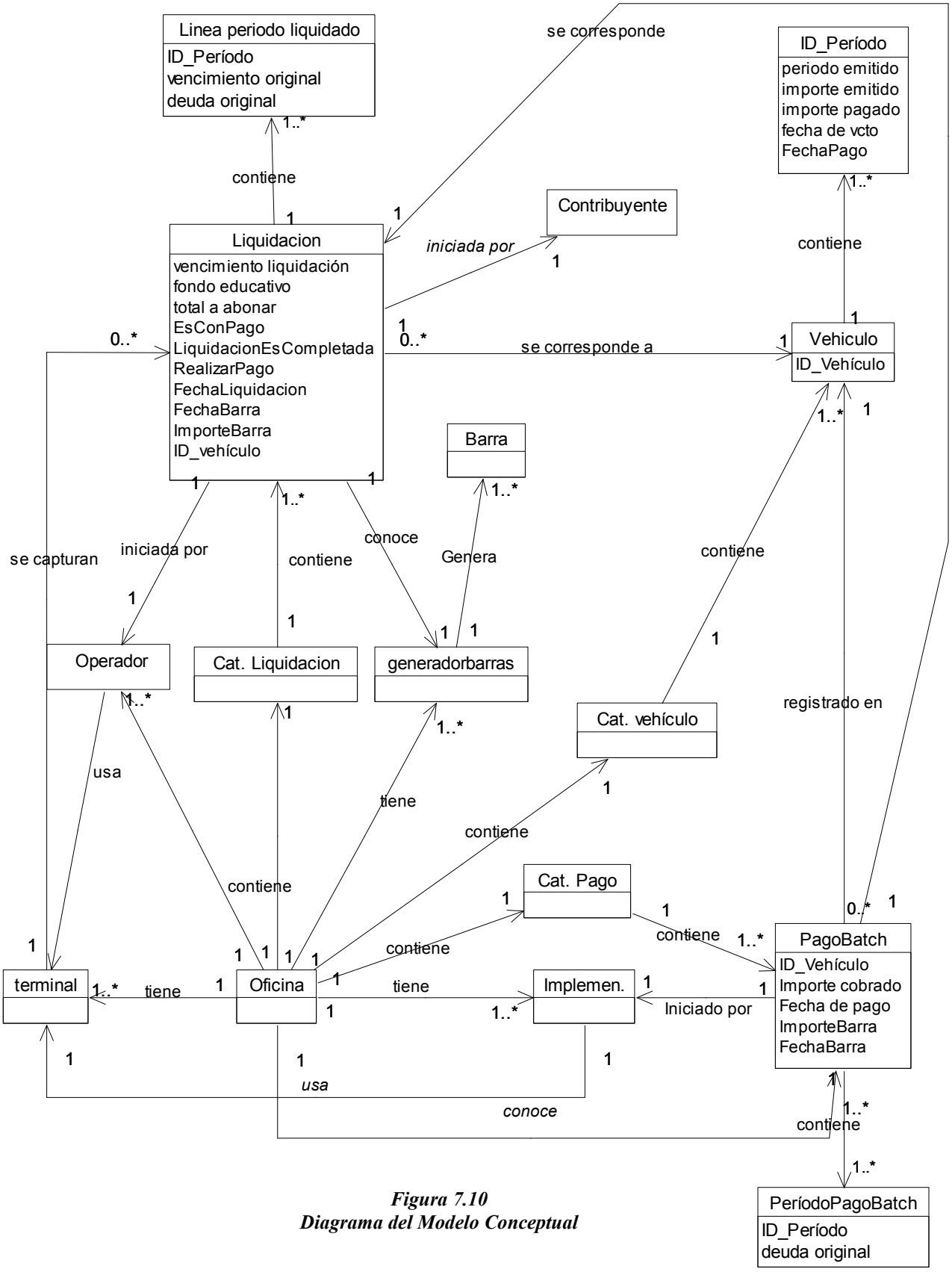


Figura 7.10
Diagrama del Modelo Conceptual

Capítulo 8 -Revisión del Caso de Uso ImputarPagos.

8.1- Introducción.

Cuando intentamos realizar el diagrama de colaboración iniciado por el evento de sistema ImputarPago(datos), nos encontramos con un diagrama que no nos resultó satisfactorio. Como consecuencia de los modelos generados previamente, logramos que un único diagrama de colaboración realizará la siguientes tareas:

- *Generar una instancia de PagoBatch con sus correspondientes instancias de PeríodosPagosBatch.*
- *Validar e imputar el PagoBatch recién ingresado.*

Los dos puntos enunciados arriba constituyen dos funcionalidades bien específicas que no necesariamente deben ser realizadas en un mismo momento. Es decir, primero se puede cargar todo el catálogo de PagoBatch, y luego, en un proceso posterior, se puede validar e imputar.

La secuencia elegida sería:

- *Se carga los datos del archivo de pagos proveniente del Banco en el catálogo de PagoBatch como pendientes de imputación.*
- *El catálogo ya se encuentra en el sistema y contiene todos los Pagos que el banco ha mandado para ser imputados (los pagos del proceso actual, los imputados o no en procesos anteriores). Dicho catálogo se carga en localmente justo antes de generar los nuevos pagos pendientes de imputación.*
- *Se procesan los pagos pendientes de imputación: validándolos, e imputándolos (en caso de que la validación resulte correcta) o marcándolos como erróneos (en caso de que la validación resulte incorrecta).*

En base a lo expuesto y revisando los casos de uso vistos hasta ahora, nos damos cuenta de que hemos omitido un caso de uso que llamaremos CargarPagosImputar y de que necesitamos modificar el caso de uso ImputarPagosBatch.

8.2- Acerca de la organización de este capítulo.

En este capítulo, modificaremos el caso de uso ImputarPagosBatch y escribiremos el nuevo caso de uso CargarPagosImputar.

A continuación, corregiremos el diagrama conceptual, el diagrama de secuencias (donde descubrimos las operaciones del sistema), y reveremos los contratos del sistema.

Esperamos, de esta manera, poder llegar a un diagrama de colaboración satisfactorio.

8.3- Diagramas de Caso de uso

8.3.1- Creación del diagrama de Caso de uso CargarPagosImputar.

Actores: *Entidad Bancaria(iniciador), Implementador*
Propósito: *Carga el pago en el Catálogo de pago.*
Descripción: *La entidad bancaria suministra la información necesaria para el asentamiento de pagos realizados en su institución. El operador toma la información y la carga en el catálogo de pagos para su posterior imputación.*
Tipo: *Primario y esencial*

Referencias cruzadas: *Funciones: 2.5 y 2.6*
Caso de Uso:ImputarPago

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando la entidad bancaria envía los datos de pagos realizados en su institución.	
2-El implementador del pedido da la orden de cargar los pagos como pendientes de imputación	*3- Por cada pago ingresado:* - *Se cargará los datos enviados en el catálogo de pagos.* - *Los mismos son: la identificación del vehículo, el importe de la barra, el importe cobrado, la fecha de la barra, la fecha de pago y todas las identificaciones de períodos e importe originales que se pagaron.* - *Se marca el pago como pendiente de imputación, poniendo una "I" en el estado de pago.*
	4- Se lista los totales de la carga.
5- El implementador lee los totales.	

Nota:

Asumiremos que los pagos vienen en el formato de entrada correcto (ejemplo: los números son números y no alfabéticos, etc.).

Por otro lado, tener en cuenta que cada Pago debe ser enviado de manera que se corresponda con una liquidación. Es decir, si un contribuyente pago períodos adeudados de su vehículo con 2 liquidaciones diferentes; entonces el banco enviará 2 pagos, uno por cada liquidación realizada. Esto es por motivos de validación de la imputación. Si se recibiera la información en un solo pago, entonces no se imputarán los períodos ya que resultarán erróneos.

8.3.2- Modificación del caso de uso ImputarPagosBatch

Actores: *Implementador (iniciador)*
Propósito: *Realizar asentamiento de pagos*
Descripción: *El implementador dá la orden de ejecución y se produce la imputación o el marcado como erróneo de los pagos que se encuentran en el catálogo de pagos y que están pendientes de imputación.*
Tipo: *Primario y esencial*
Referencias cruzadas: *Funciones: 2.5 y 2.6*
Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el implementador dá la orden de que se proceda a la imputación de los pagos.	*2 – Se tomarán todos los pagos cuyo estado de pago indique que se encuentra pendientes de imputación* *3- Se realiza la validación de los pagos:* *3.1- Reconstruyendo el supuesto cobrado con el coeficiente de actualización (desde la fecha de barra a la fecha de pago) y el importe de la barra y verificando su igualdad con el importe cobrado en los datos del pago.* *3.2- Obteniendo la liquidación asociada al pago ingresado (se encuentran en el catálogo de liquidaciones). Esta liquidación posee las siguientes características:* - *La identificación del dominio, el importe de barra y la fecha de barra es igual a la identificación de dominio, el importe y la fecha de barra del pago ingresados vía banco.* - *La fecha de liquidación es menor o igual a la fecha de pago.* - *Todos los períodos e importes liquidados son iguales a los que contiene el pago ingresados vía banco.* - *No posee pago anterior asociado.*

	4- Si durante la validación llegamos a la conclusión de que el pago estaba correcto, entonces se imputará: - *Sumando el importe emitido de cada período ingresados al importe pagado del vehículo para ese período.* - *Modificando el estado del pago para indicar que ya se encuentra imputado.* - *Modificando la liquidación asociada para indicar que ya ingresó el pago.* - *Sumando la información sobre el pago ingresado en el Sistema de Entrantes (se debe asentar el dinero que el banco le debe a la entidad recaudadora)* *5- Si durante la validación llegamos a la conclusión de que el pago estaba incorrecto, entonces se modificará el estado del pago en el catálogo para indicar que ya se encuentra erróneo.* *6-Imprime totales sobre los pagos imputados y los marcados como erróneos*
7- El implementador le da los datos de la imputación a la entidad bancaria (Total de toda la imputación, Dominios Imputados.)	

Curso alternativo:

2- Si no existen pagos pendientes de imputación, entonces se informa de la situación y se termina el proceso.

8.4- Revisando el modelo conceptual

Estudiando los casos de uso de las secciones anteriores de este capítulo, vemos que no se han incorporado nuevos conceptos a la aplicación en estudio. Lo nuevo que ha aparecido es la necesidad de marcar cada PagoBatch como pendiente de imputación o como imputado o como con error. Esto lo haremos mediante la incorporación de un nuevo atributo en el concepto PagoBatch que indique el estado del pago.
El modelo conceptual queda de la siguiente manera:

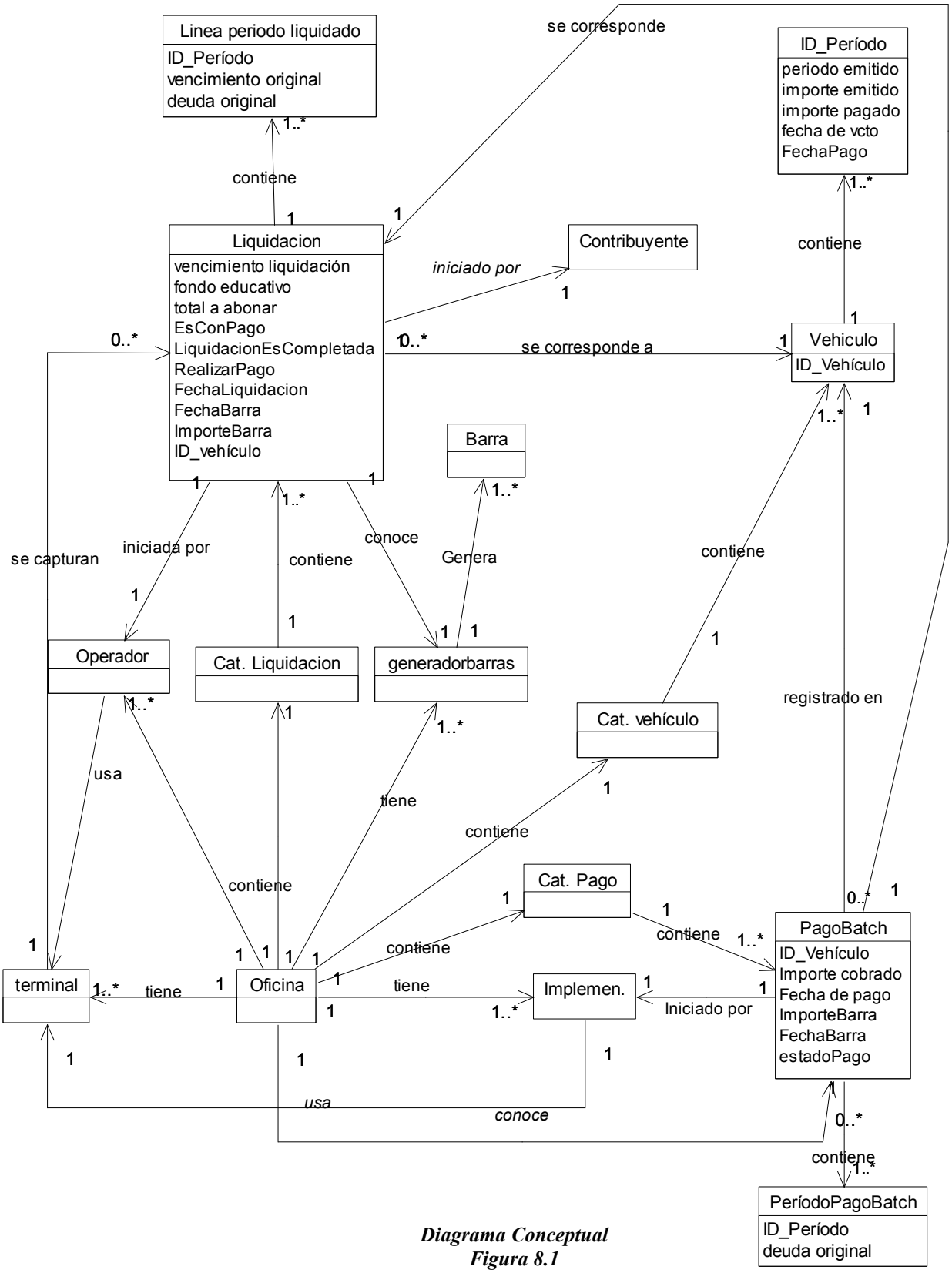
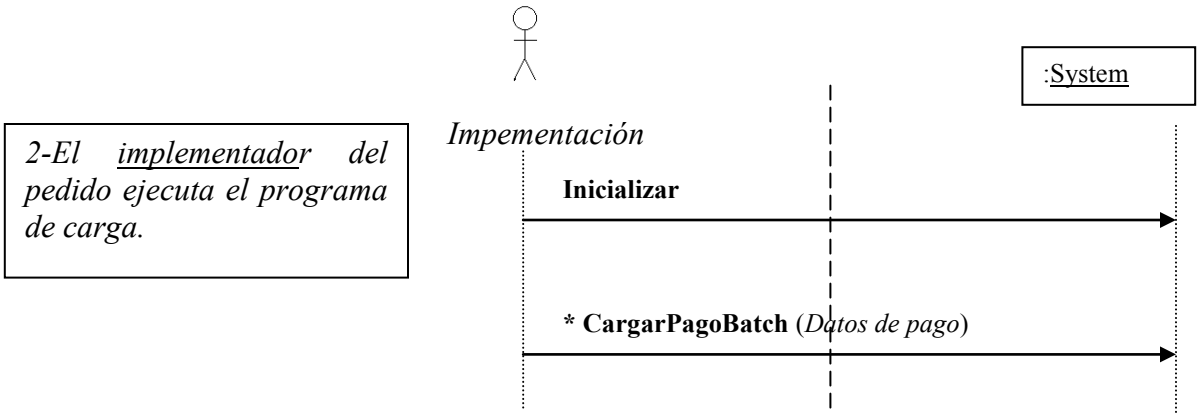


Diagrama Conceptual
Figura 8.1

8.5- Revisando los diagramas de secuencias.

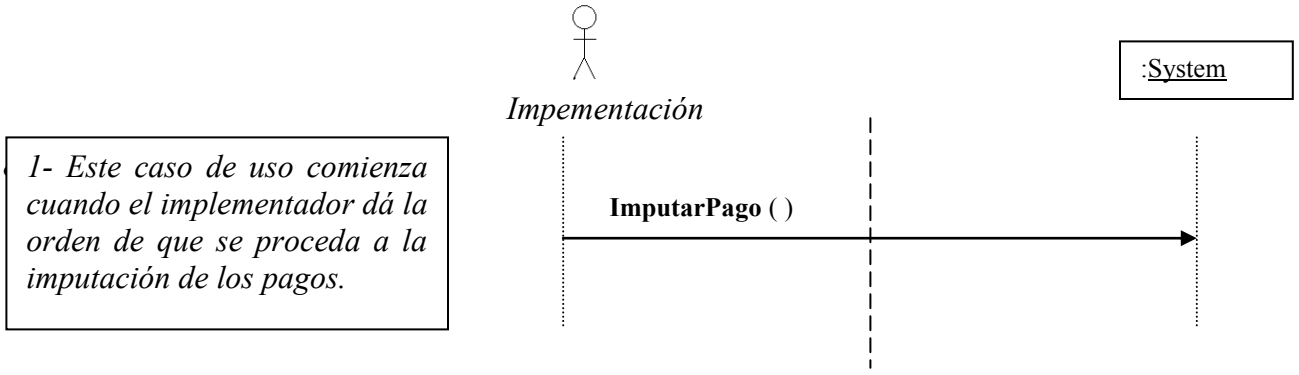
8.5.1- Creación del diagrama de secuencias para CargaPagoBatch.



8.5.2- Revisando diagrama de secuencias de ImputarPagosBatch

Mirando el diagrama de secuencias ImputarPagosBatch del capítulo 4, vemos que lo único cambiará es que ya no serán necesarias los parámetros de entrada de la operación del sistema.

De esta manera nos quedará de la siguiente forma:



8.5.3- Operaciones del sistema.

1- EntrarPeriodo (identificacióndominio, período)
2- TerminarLiquidación( )
3- RealizarPago( )
4- Inicializar()
5- CargarPagoBatch(Datos de Pagos)
6- ImputarPago ()

Hemos puesto con letra negrita las operaciones de sistema nuevas (puntos 4 y 5) y la que sufre modificaciones (punto 6). En las secciones subsiguientes elaboraremos los contratos y diagrama de colaboración de estas operaciones de sistema ya que las de los puntos 1 a 3 no sufren modificaciones.

8.6- Revisión de los contratos

8.6.1- Creación del Contrato Inicializar.

Nombre:	***Inicializar***
Responsabilidades:	*Inicialización del sistema previa a la carga de pagos a imputar*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 2.5 y 2.6.* *Casos de Uso: CargarPago*

Post-condiciones:

- *El Catálogo de PagoBatch es creado.*
- *La Oficina fue asociada al Catálogo de PagoBatch.*
- *Para cada Pago existente en la base de datos:*
  - *Un nuevo PagoBatch fue creado.*
  - *El PagoBatch.identificadorVehículo fue fijado.*
  - *El PagoBatch.importeCobrado fue fijado.*
  - *El PagoBatch.fechaPago fue fijado.*
  - *El PagoBatch.fechaBarra fue fijado.*
  - *El PagoBatch.importeBarra fue fijado.*
  - *El PagoBatch.EstadoPago fue fijado.*
  - *Para cada período que posee el pago:*
    - *Un PeríodosPagoBatch fueron creados*
    - *El PagoBatch fue asociado a cada uno de los PeríodosPagoBatch*
    - *Los PeríodoPagoBatch.PeríodoImputar fueron fijados para todos los PeríodosPagoBatch.*
    - *Los PeríodoPagoBatch.DeudaOriginal fueron fijados para todos los PeríodosPagoBatch.*

8.6.2- Creación del Contrato CargarPagoBatch.

Nombre:	*CargarPago* *(ID_Vehículo, FechaBarra, ImporteBarra, ImporteCobrado, FechaPago, Líneas conteniendo el ID_Vehículo y la deudaOriginal)*
Responsabilidades:	*Cargar los datos de los pagos provenientes del banco como pendiente de imputación.*
Tipo:	*Sistema.*
Referencias Cruzadas:	*Funciones del sistema: 2.5 y 2.6* *Casos de Uso: ImputarPago*
Pre-condiciones:	*El catálogo de Pago ya se encuentra cargado en el sistema.*

Post-condiciones:

- *Un nuevo PagoBatch fue creado.*
- *El Catálogo de PagoBatch fue asociado la PagoBatch.*
- *El PagoBatch fue asociado a la Terminal.*
- *El PagoBatch.identificadorVehículo fue fijado.*
- *El PagoBatch.importeCobrado fue fijado.*
- *El PagoBatch.fechaPago fue fijado.*
- *El PagoBatch.fechaBarra fue fijado.*
- *El PagoBatch.importeBarra fue fijado.*
- *El PagoBatch.EstadoPago fue fijado en "I".*
- *Para cada período que posee el Pago:*
  - *Nuevos PeríodosPagoBatch fueron creados*
  - *El PagoBatch fue asociado a cada uno de los PeríodosPagoBatch*
  - *Los PeríodoPagoBatch.PeríodoImputar fueron fijados para todos los PeríodosPagoBatch.*
  - *Los PeríodoPagoBatch.DeudaOriginal fueron fijados para todos los PeríodosPagoBatch.*

8.6.3. Revisión del Contrato ImputarPago

Nombre:	*ImputarPago()*
Responsabilidades:	*Registrar en el sistema el pago realizados por contribuyentes a través del banco.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 2.5 y 2.6.* *Casos de Uso: CargarPago*
Pre-condiciones:	*El catálogo de Pago ya se encuentra cargado con los datos pendientes de imputación.*

Post-condiciones:

- *Para cada PagoBatch en donde:*
  - *el estado de pago estaba en "I"*
  - *el importecobrado estaba correcto*

  entonces Liquidaciones se asociaron a PagoBatch.
- *Para cada PagoBatch en donde:*
  - *el estado de pago estaba en "I"*
  - *el importecobrado estaba correcto*
  - *se encontró liquidación con igual identificación de dominio*

  entonces:
  - *PeríodoVehículo.ImportePagado fue modificado para cada período que se pagó.*
  - *PeríodoVehículo.FechaPago fue modificado para cada período que se pagó.*

- *Liquidación.RealizarPago fue fijado en True.*
- *PagoBatch.estadoDePago fue fijado en "P".*

- *Para cada PagoBatch en donde el estado de pago estaba en "I" y donde:*
  - *el importecobrado no estaba correcto*

  o
  - *no se encontró liquidación asociada a Pago*

entonces PagoBatch.EstadoPago fue fijado en "E".

8.7- Diagramas de Colaboración.

Crearemos los diagramas de colaboración de las operaciones de sistema "Inicializar", "CargarPago" e "ImputarPago". Utilizaremos para su realización el diagrama conceptual y los contratos mostrados en las secciones anteriores de este capítulo.

8.7.1- Diagrama de Colaboración de Inicializar.

Realizamos la operación de Inicialización del Catalogo de PagoBatch en este lugar y no en el Startup del sistema, por considerar que dicho catálogo es utilizado solamente por el caso de uso ImputarPagosBatch y CargarPagoBatch. Luego, dado que no posee un uso global (como los otros catálogos) no consideramos que su inicialización requiera ser realizada en el Startup de la aplicación sino en el lugar donde se lo usa.

La inicialización del Catálogo de PagoBatch implica:
- La creación del Catálogo de PagoBatch.
- La carga de los PagosBatch que se encuentran en la base de datos.

En la aplicación real, las instancias de PagoBatch residirán en un medio de almacenaje persistente tal como una base de datos orientada a objetos o relacional. Si poseen pocos objetos, estos pueden ser cargados en la memoria del computador. Sin embargo, si hay muchos su carga consumirá mucho tiempo y memoria. Alternativamente las instancias serán cargadas en memoria cuando se las requiera.

El diseño de cómo se carga los objetos en memoria de manera dinámica es un problema que relegaremos a ciclos posteriores, para no dificultar en demasía nuestro desarrollo. En este ciclo supondremos que todas las instancias de PagoBatch pueden ser creadas en memoria por el objeto CatálogoPagoBatch.

El diagrama de colaboración tendría el siguiente formato:

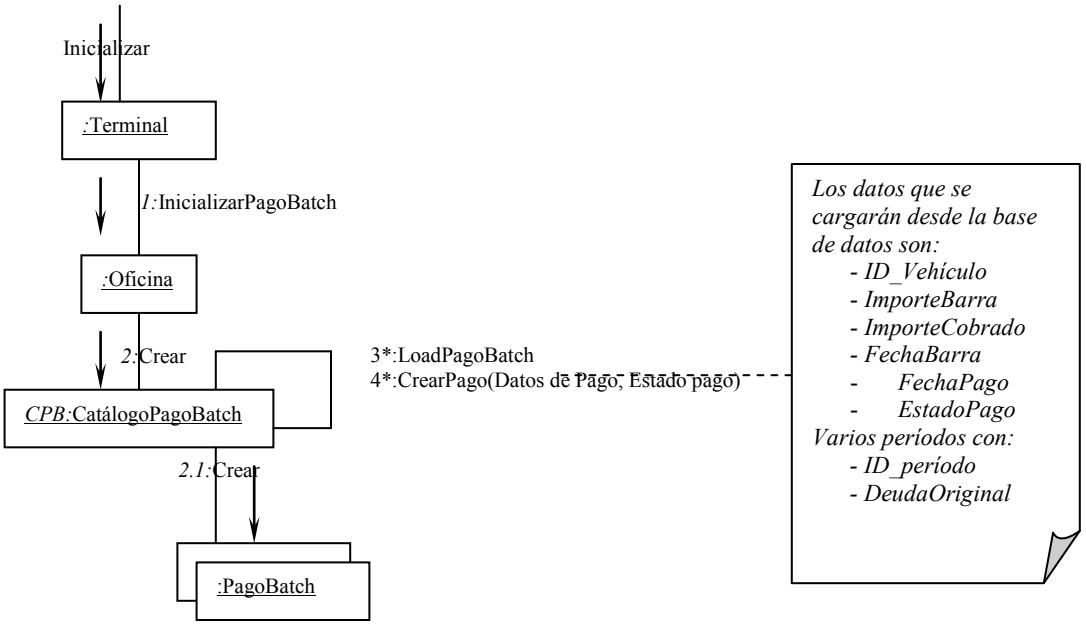


Diagrama de Colaboración de Inicializar
Figura 8.2

Cuando Terminal envía el mensaje "InicializarPagoBatch" a Oficina esta tendrá la responsabilidad, por patrón creador y dado que ella mantiene los catálogos de la aplicación, de crear el Catálogo de PagoBatch.

Catálogo de PagoBatch tendrá la responsabilidad por experto de:

- *Crear la colección de PagosBatch que él mantiene.*
- *Solicitar los datos del PagoBatch que serán cargados.*
- *Cargar en memoria los pagos con todos los datos recibidos en el punto anterior (hemos separado el estado del pago respecto al resto de los atributos para lograr la reusabilidad de la operación CrearPago).*

Postergaremos el estudio de la operación de CrearPago por su potencial reusabilidad, hasta después de examinado el diagrama de colaboración de CargarPagoBatch. Tanto la operación de sistema Inicializar como CargarPago poseen la función de crear PagoBatch. Una crea un pago con los datos provenientes de la base de datos(inicializar), y la otra carga los pagos con los datos provenientes del banco vía Terminal y Oficina (CargarPago).

8.7.2- Diagrama de Colaboración de CargarPagoBatch.

Cuando Terminal envía el mensaje "CargarPagoBatch" a Oficina, esta tendrá la responsabilidad, por patrón experto, de solicitarle al Catálogo de PagoBatch la creación de una nueva instancia de PagoBatch. Para su creación, requerirá que se le envíe, vía parámetro los datos del pago y el estado del mismo (en este caso será siempre "I").

El diagrama de colaboración tendrá el siguiente formato:

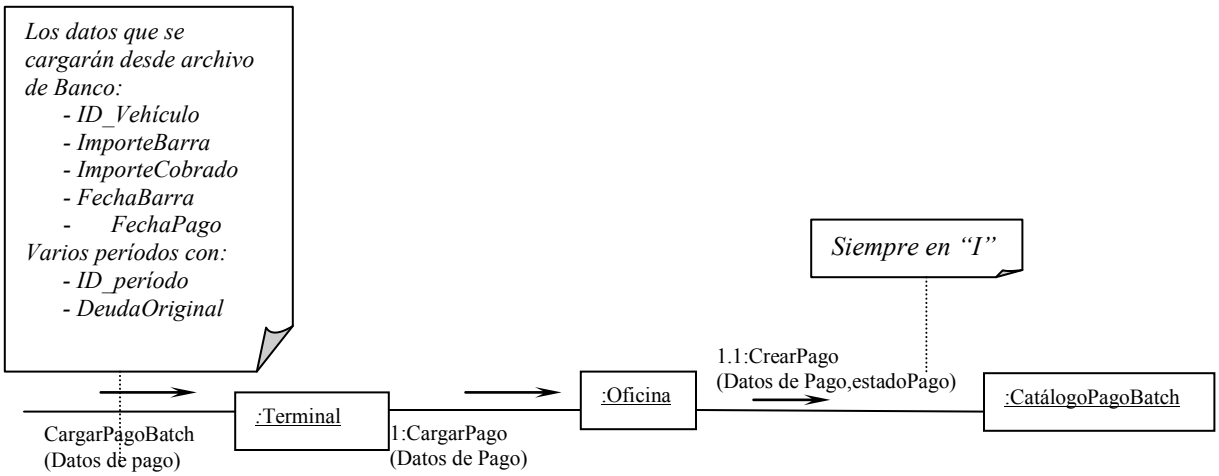


Diagrama de Colaboración de CargarPagoBatch
Figura 8.3

8.7.3- Diagrama de colaboración de CrearPago

Esta operación será utiliza en las operaciones de sistema "Inicializar" y "CargarPagoBatch". Los datos de que se ingresan serán enviados por la base de datos o por la Oficina dependiendo de quien envía el mensaje.

En primer lugar, consideremos las siguientes post-condiciones existentes tanto en el contrato de sistema "Inicializar" y "CargarPagoBatch":

- *Un nuevo PagoBatch fue creado.*
- *El Catálogo de PagoBatch fue asociado la PagoBatch.*
- *El PagoBatch.identificadorVehículo fue fijado.*
- *El PagoBatch.importeCobrado fue fijado.*
- *El PagoBatch.fechaPago fue fijado.*
- *El PagoBatch.fechaBarra fue fijado.*
- *El PagoBatch.importeBarra fue fijado.*
- *El PagoBatch.EstadoPago fue fijado.*

La primer post-condición indica la responsabilidad de creación del PagoBatch. Mirando el diagrama conceptual, vemos que el Catálogo de PagoBatch contiene PagosBatch, luego por modelo creador, es la clase indicada para tener la responsabilidad de crear una nueva instancia de PagoBatch y de sumarlo a la colección que él mantiene. Se utilizará los datos ingresados por parámetro para la inicialización de los atributos del nuevo PagoBatch.

Ahora consideremos el siguiente grupo de post-condiciones (también existentes en los contratos de sistema"CargarPagoBatch" e "Inicializar")

Consideremos ahora el segundo grupo de post-condiciones:

- *Nuevos PeríodosPagoBatch fueron creados*
- *El PagoBatch fue asociado a cada uno de los PeríodosPagoBatch*

- Los PeríodoPagoBatch.PeríodoImputar fueron fijados para todos los PeríodosPagoBatch.

- Los PeríodoPagoBatch.DeudaOriginalr fueron fijados para todos los PeríodosPagoBatch.

Lo primero que se notamos en estas post-condiciones es la responsabilidad de creación de los PeríodosPagoBatch. Se crearán tantos PeríodosPagoBatch como los ingresados en los datos de entrada. El modelo conceptual nos muestra que PagoBatch contiene los PeríodosPagoBatch, entonces es la clase indicada (por modelo creador) para crearlos y sumarlos a la colección (ya que PagoBatch mantiene la colección que contiene a los períodos).

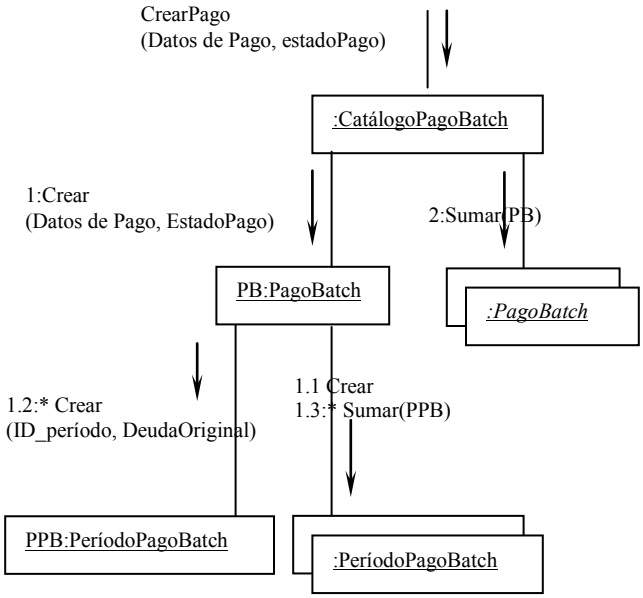


Diagrama de Colaboración de CargarPago de Catálogo de PagoBatch
Figura 8.4

8.7.4. Diagrama de Colaboración de ImputarPagos

8.7.4.1- Iteración de cada pago

Cuando Terminal recibe el mensaje de imputación de pagos, ella enviará sucesivos mensajes sí misma indicando el procesamiento de cada pago que se encuentra en el Catálogo de PagoBatch y que está pendiente de imputación. El diagrama inicial tiene de esta manera el siguiente formato:

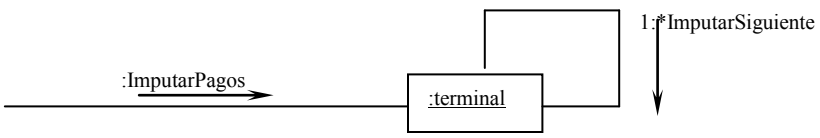


Diagrama de colaboración de ImputarPagos
Figura 8.5

8.7.4.2- Imputación de un pago

Dado que en la oficina se encuentra toda la información necesaria para el proceso de imputación (Catálogo de PagoBatch, Catálogo de Liquidación y Catálogo de Vehículo), por experto es la clase indicada para tener la responsabilidad de imputar un pago, y puesto que existe una conexión entre la Terminal y Oficina (que se realiza en el momento de Start up del equipamiento), Terminal puede enviarle a la Oficina un mensaje para solicitarle que procese el próximo pago pendiente de imputación.

Lo primero que tenemos que hacer es obtener el próximo pagoBatch que este pendiente de imputación (tener en cuenta que si un pago esta pendiente de imputación entonces el atributo estado de pago de la clase PagoBatch, se encuentra en "I").

Nos preguntamos ¿Quién es responsable de obtener el próximo PagoBatch pendiente de imputación?. Dado que los PagosBatch se encuentran contenidos en un Catálogo de PagoBatch, luego, por experto, este último, es la clase candidata tener tal responsabilidad

Lo segundo que se tiene que hacer es validar el PagoBatch obtenido. Para ello realizaremos una serie de pasos:

1- *Validar el supuesto cobrado.*
2- *Buscar las liquidaciones realizadas para este dominio, que no tengan pago asociado.*
3- *Compararlas con el PagoBatch que estamos procesando para verificar si se corresponde con él.*

Examinemos con más detalle los puntos que hemos enunciado arriba:

1- *Para calcular el supuesto cobrado debemos comparar el atributo ImporteCobrado de PagoBatch contra la suma del ImporteBarraGral + el interés que resulta de la FechaBarraGral y la FechaPago. Notemos que todo el cálculo se realiza en base a atributos de la instancia de la clase PagoBatch, luego por experto, ella es la clase responsable de realizar tal validación. Teniendo en cuenta que existe una conexión entre Oficina y PagoBatch, entonces Oficina le solicitará a PagoBatch que valide el importe Cobrado vía mensaje.*
2- *Mirando el modelo conceptual, vemos que el Catálogo de Liquidaciones contiene todas las liquidaciones realizadas y que se encuentra en la Oficina. Por experto, podemos concluir dos cosas:*
    - *Catálogo de Liquidaciones tiene la responsabilidad de obtener (si es que lo encontrase) la próxima liquidación que se corresponda al vehículo cuyos pagos se desean imputar y que no tenga un pago ya asociada (atributo realizarPago de la liquidación en false). Esta búsqueda se realizará tantas veces hasta que se encuentra la liquidación asociada al PagoBatch o hasta que no haya más Liquidaciones para el Vehículo (en cuyo caso deduciremos que el PagoBatch ingresado es erróneo).*
    - *Oficina será responsable de enviar el mensaje, al Catálogo de Liquidaciones para que él obtenga la próxima liquidación a realizar.*
3- *Oficina le enviará un mensaje a PagoBatch para que se compare con la instancia de liquidación encontrada. (la comparación la realizaremos en otro*

diagrama de colaboración para evitar que el diagrama en estudio se vuelva engoroso). En caso de que no existiese correspondencia entre la liquidación y el pagoBatch, entonces buscaremos otra nueva liquidación (volveremos a realizar el punto 2)

Lo tercero que debemos realizar es:

- El proceso de imputación de pago (en caso de que la validación resultó satisfactoria)

- *El proceso de marcada de error (en caso de que durante el proceso de validación hayamos concluido de que el pago era erróneo)*

Dividiremos el diagrama de colaboración y trataremos realización del pago en un diagrama subsecuente.

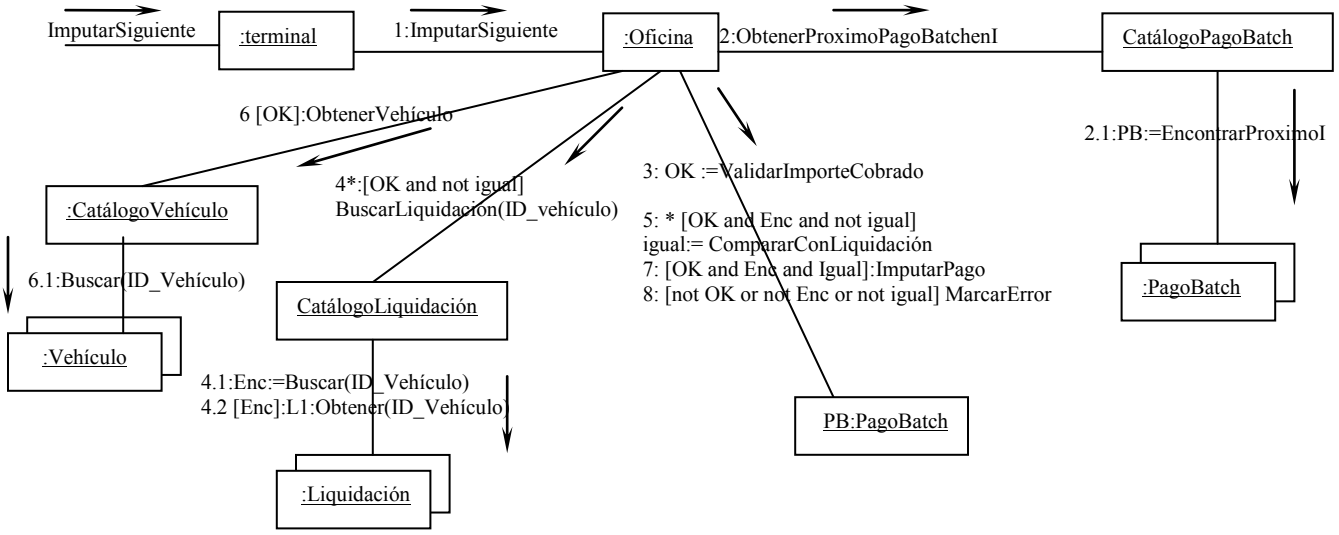


Diagrama de colaboración de ImputarSiguiente
Figura 8.6

8.7.4.3- Comparación de Liquidación y PagoBatch

Cuando el PagoBatch recibe el mensaje de compararse con la liquidación, entonces deberá:

1- *Verificar la igualdad del importe de barra general y de la fecha de la barra general. PagoBatch le solicitará a la liquidación que le devuelva los datos necesarios para realizar tal comparación; ya que ella es la clase indicada, por experto, para tener la responsabilidad de devolver los datos necesarios.*
2- *Verificar que la fecha de pago sea mayor o igual a la fecha de la liquidación. Nuevamente PagoBatch le solicitará a la liquidación que le devuelva su fecha de liquidación.*
3- *Para cada uno de los períodos de pagoBatch*
   - *Se deberá comparar el período y el Importe original contra el de la liquidación. Luego PagoBatch le solicitará a la Liquidación, uno a uno cada uno de los períodos y de los importes originales, para compararlos con sus iguales.*

El diagrama resultante es:

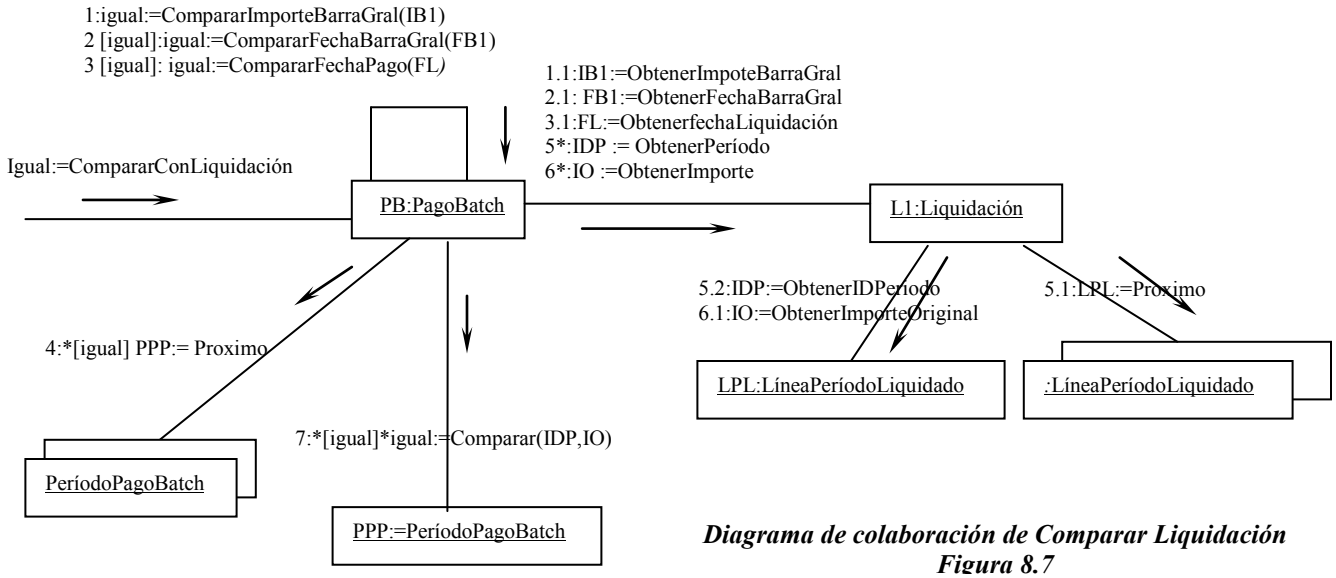


Diagrama de colaboración de Comparar Liquidación
Figura 8.7

Nota: *En caso de que en alguna comparación los valores resultaran distintos, entonces el proceso terminará.*

8.7.4.4- Proceso de Imputación de Pago.

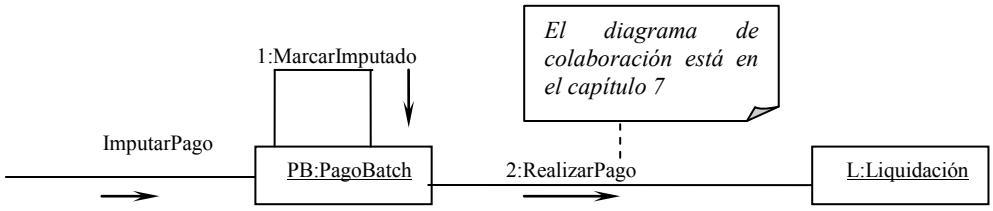


Diagrama de colaboración de ImputarPago
Figura 8.8

Los mensajes "MarcarImputado" y "MarcarErroneo" no lo hemos ampliado porque lo único que se realizan es un cambio de atributo (el atributo EstadodePago del PagoBatch pasará a estar de "I" a "P" o "E" respectivamente).

Con respecto al mensaje realizarPago no lo hemos ampliado en este diagrama de colaboración, ya que es un mensaje del diagrama de colaboración de RealizarPago estudiado en el capítulo 7. Lo que haremos será, sacarlo de dicho diagrama para lograr la reusabilidad del mismo en ambos esquemas (ver capítulo 9).

Capítulo 9
Diagramas de colaboración finales y Conexión con la interfase del usuario.

9.1- Acerca de la organización del capítulo.

Antes de realizar el diagrama de clases, creemos conveniente mostrar los diagramas de colaboración obtenidos durante los capítulos 7 y 8. Mostraremos primero los diagramas de colaboración que surgen de los eventos del sistema (ver los diagramas de secuencia del capítulo 5 y 8) y luego mostraremos los demás diagramas.

Por último, realizaremos una muestra de cómo se asocia el nivel de presentación con el nivel de dominio.

9.2- Diagramas de colaboración finales

9.2.1- Diagrama de colaboración de los eventos del sistema.

9.2.1.1- Diagrama de colaboración de EntrarPeríodo.

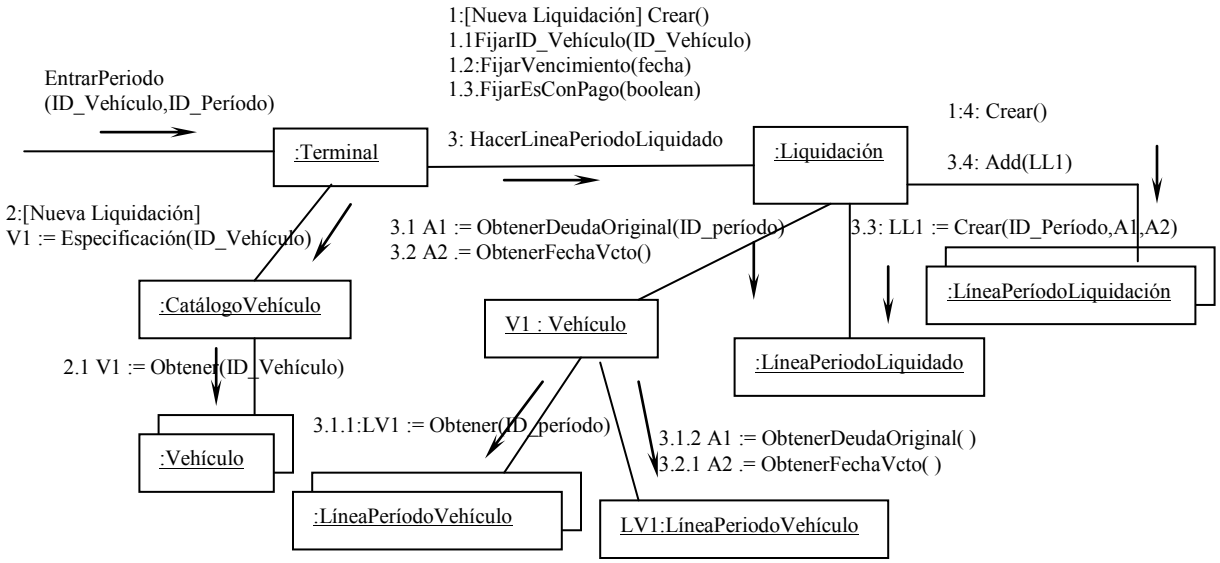


Figura 9.1

9.2.1.2- Diagrama de colaboración de TerminarLiquidación

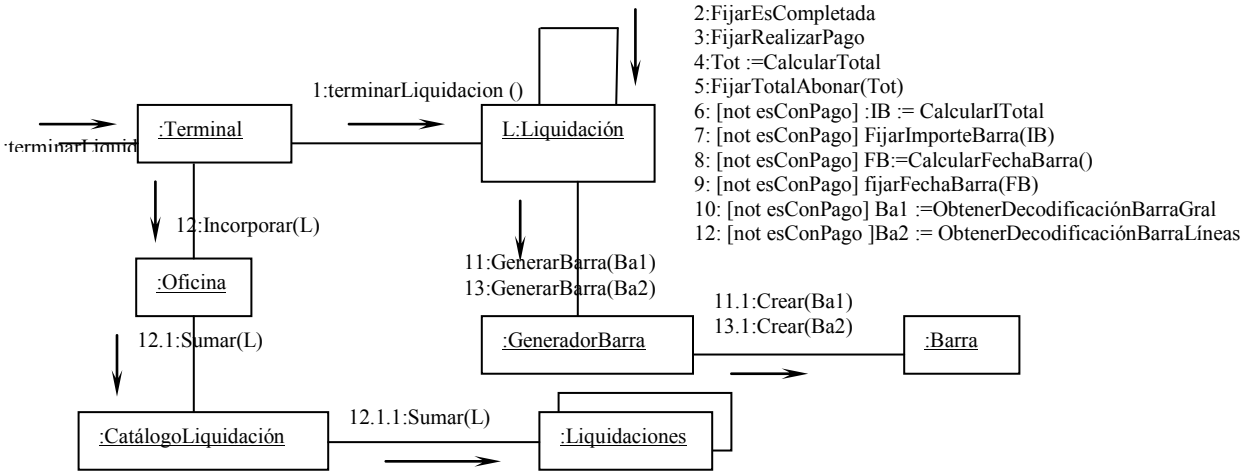


Figura 9.2

9.2.1.3- Diagrama de colaboración de RealizarPago.

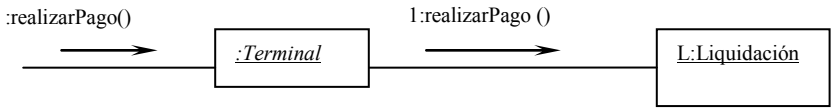


Figura 9.3

***Nota**: El mensaje "realizarPago()" es utilizado en este diagrama de colaboración y en el de "ImputarPagoBatch".*

9.2.1.4- Diagrama de colaboración de Inicializar.

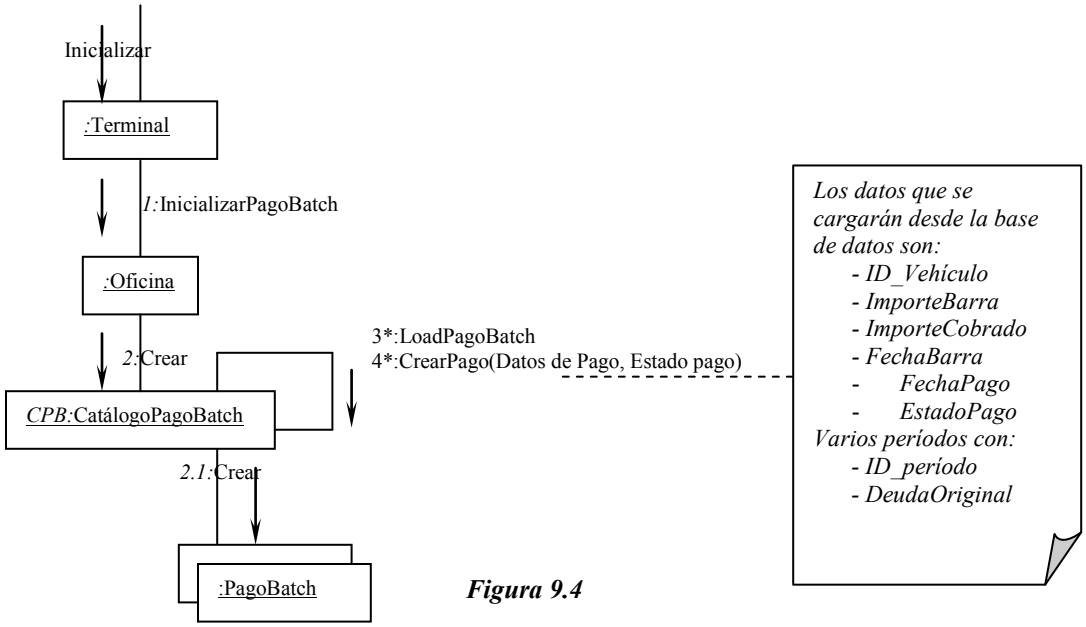


Figura 9.4

9.2.1.5- Diagrama de colaboración de CargarPagoBatch.

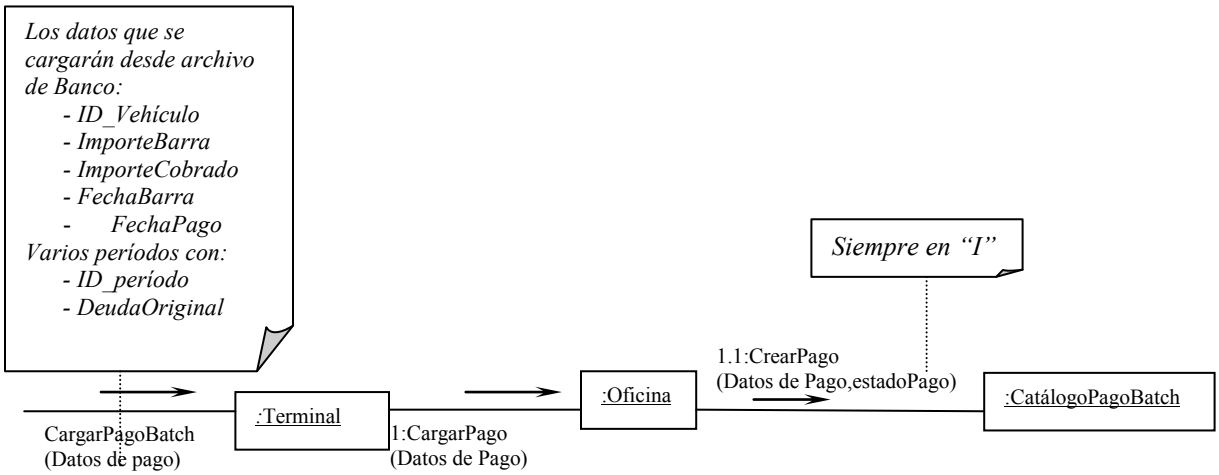


Figura 9.5

9.2.1.6- Diagrama de colaboración de ImputarPagoBatch

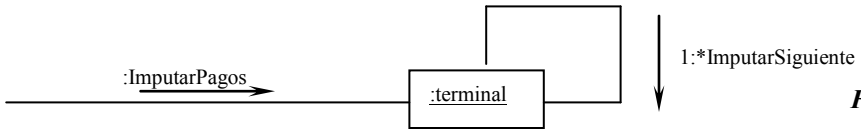


Figura 9.6

9.2.2- Demás Diagrama de colaboración

9.2.2.1- Diagrama de colaboración de CalcularTotal

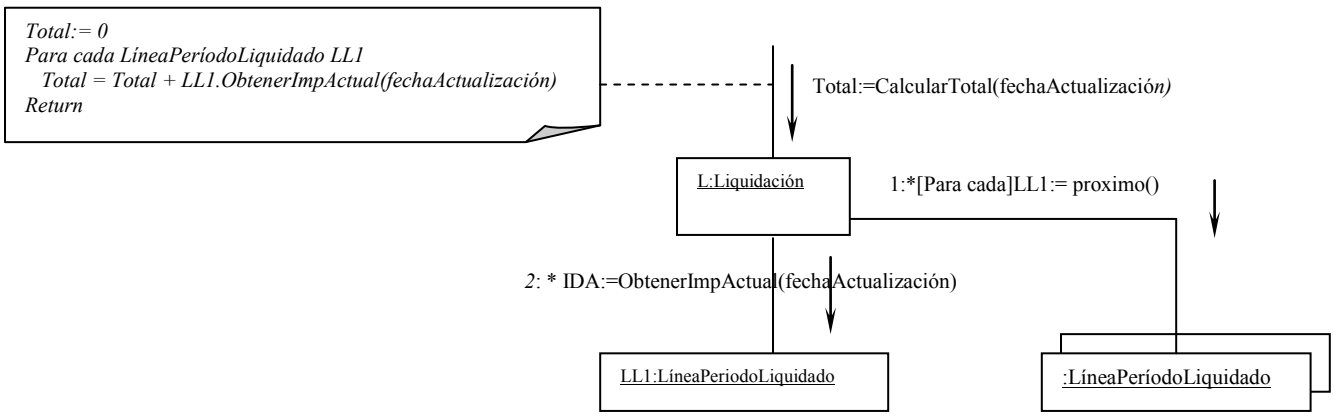


Figura 9.7

Nota:

Esta operación se utiliza en el diagrama de colaboración "terminarLiquidar".

9.2.2.2- Diagrama de colaboración de ObtenerDecodificaciónBarraLínea

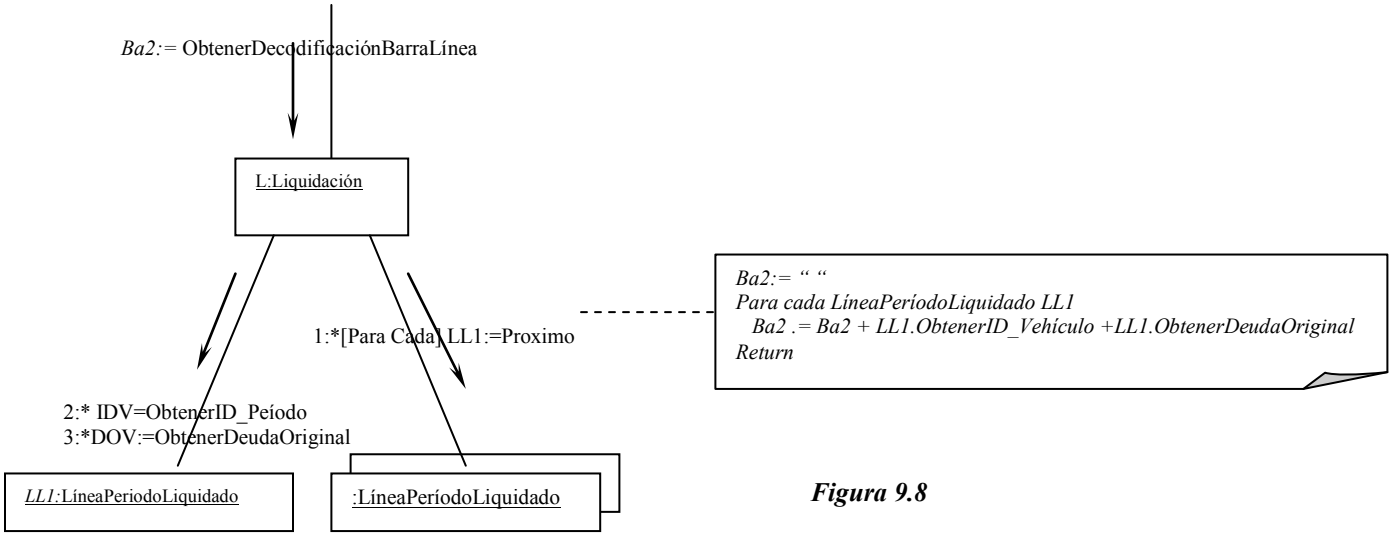


Figura 9.8

Nota:

Esta operación se utiliza en el diagrama de colaboración terminarLiquidar.

9.2.2.3- Diagrama de colaboración de RealizarPago

Hemos dividido este diagrama ya que cuando examinamos el diagrama de colaboración de ImputarPagos nos dimos cuenta que en realidad una vez que realizamos la validación del pagoBatch y estamos seguros que esta correctamente ingresado, la forma de asentamiento del pago es la misma que la realizada cuando hacemos un pago online. Luego la operación de sistema RealizarPago se hará no solo cuando se realiza un pago en forma on-line (el contribuyente paga en el momento, en la oficina de operación), sino también cuando se hace un pago en forma Batch (el contribuyente se lleva la liquidación de la oficina de rentas, va luego al banco a pagarla y es esta entidad quien reporta el pago).

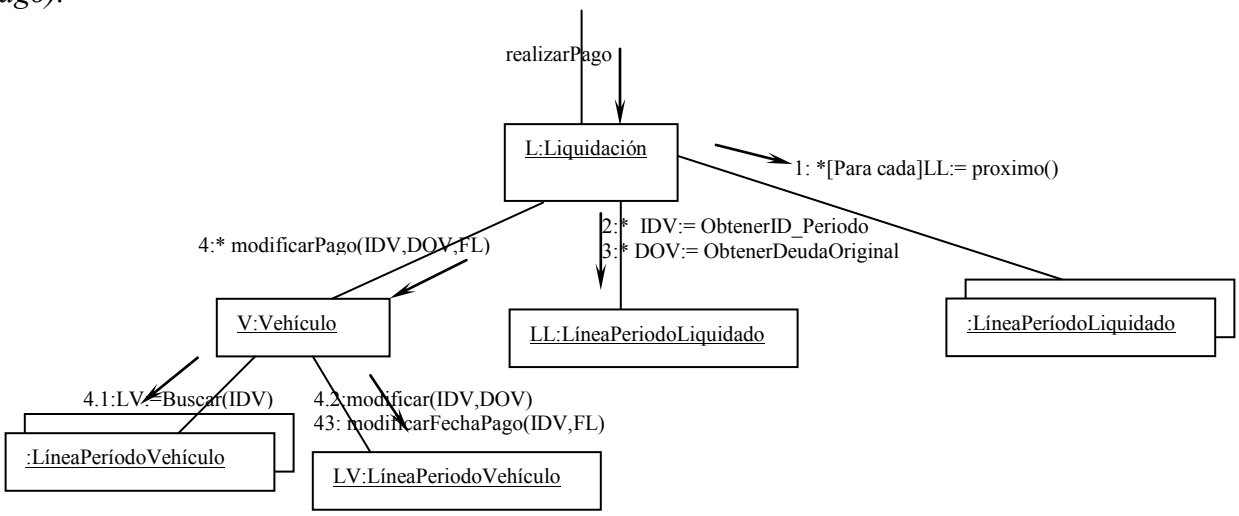


Figura 9.9

9.2.2.4- Diagrama de colaboración de CargarPago

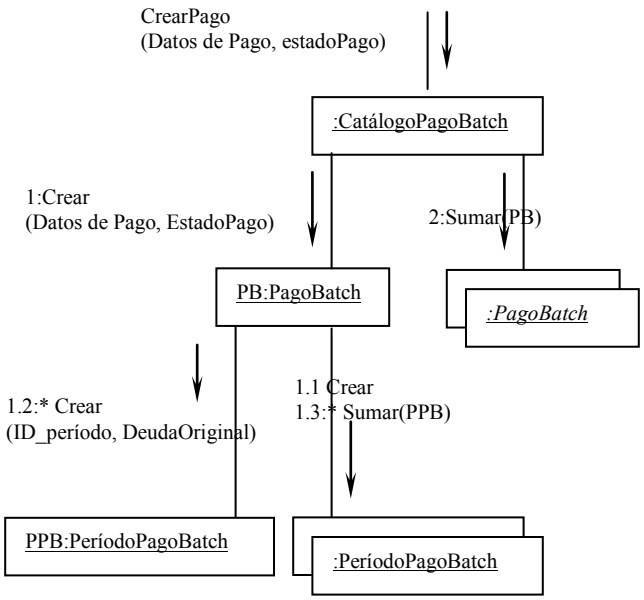


Figura 9.10

Nota:

Esta operación se utiliza en los diagramas de colaboración Inicializar y CargarPagoBach.

9.2.2.5- Diagrama de colaboración de ImputarSiguiente

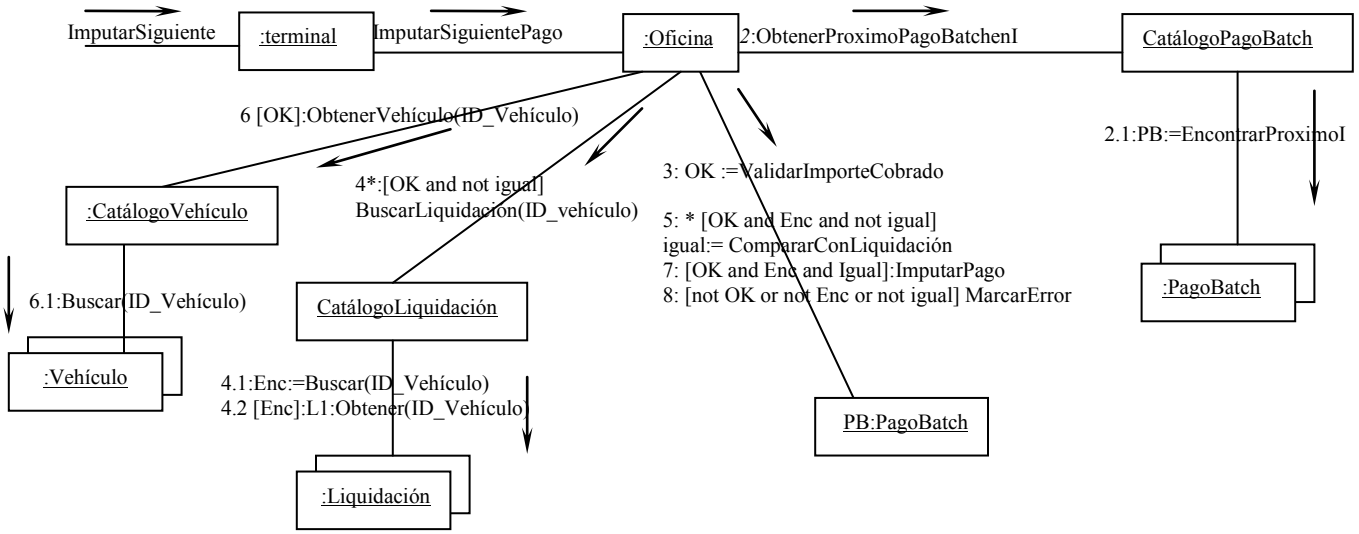


Figura 9.11

9.2.2.6- Diagrama de colaboración de CompararConLiquidación

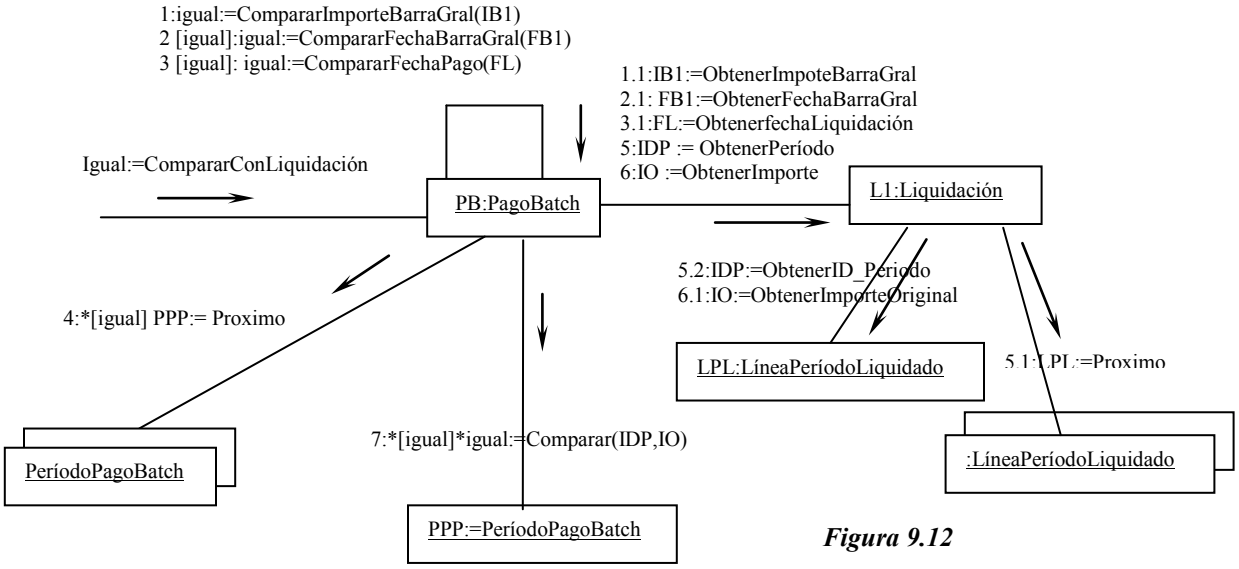


Figura 9.12

9.2.2.7- Diagrama de colaboración de ImputarPago

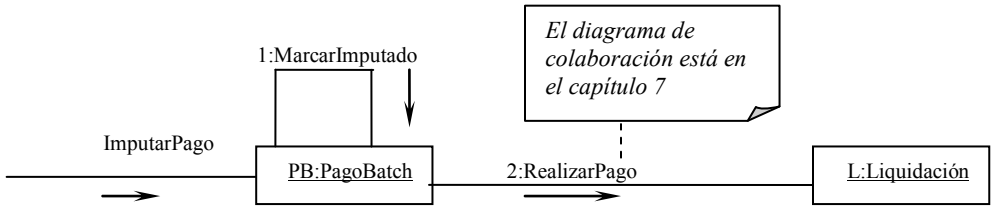


Figura 9.13

9.3- Conectando el nivel de Presentación al Nivel de Dominio.

En todos lo desarrollado hasta ahora, hemos enfatizado los objetos del dominio del problema (tal como la "Liquidación", "PagoBatch", etc.), puesto que ellos definen los conceptos y el comportamiento de la aplicación. Sin embargo un sistema está compuesto por varios subsistemas, de los cuales los objetos del dominio son uno y la interfase de usuario es otro. Un sistema de información típico debe conectarse con la interfase del usuario o nivel de presentación del sistema.

Según el patrón de "separación de modelo/vista" se debe definir a las clases del dominio (modelo) para que ellas no tengan acoplamiento directo o visibilidad o tener conocimiento de las clases de la presentación tal como ventanas o applet o reportes

(vistas). De esta manera las funcionalidades y los datos de la aplicación son mantenidos por las clases del dominio, no por las clases del nivel de presentación; logrando reusabilidad en diferentes interfases gráficas.

Sin embargo, si bien una clase del dominio no debe poseer visibilidad directa a una clase de presentación, la visibilidad indirecta es aceptable. Una forma de lograrlo, es definir una clase coordinadora de la aplicación, la cual es responsable de mediar entre el nivel de interfase y de dominio de la aplicación.

Algunas de las responsabilidades de la clase coordinadora serán:

- *Manejar la información entre el dominio del problema y la interfase del usuario.*
- *Responder a eventos de la interfase.*
- *Abrir ventanas que muestran información de los objetos del dominio*
- *Manejar transacciones tales como commit o rollback.*

Supongamos que tenemos una clase "appletTerminal" como clase coordinadora y otra clase "VistaLiquidación" en la capa presentación conectada a la ventana de liquidación. Una vez que la "appletTerminal" se haya conectado con la clase Terminal (del dominio de la aplicación), ella le enviará mensajes tales como "entrarPeríodo" o "terminarLiquidación" y tendrá visibilidad a la clase Liquidación para que se pueda mostrar en la pantalla el total a abonar. Por otra parte, la clase "appletTerminal" recibirá mensajes de la clase "VistaLiquidación" (se encuentra en el nivel de presentación y está conectada a la ventana). Se establece entonces una conexión indirecta entre las clases del dominio y las de la presentación. El esquema será mostrado en la figura 9.14.

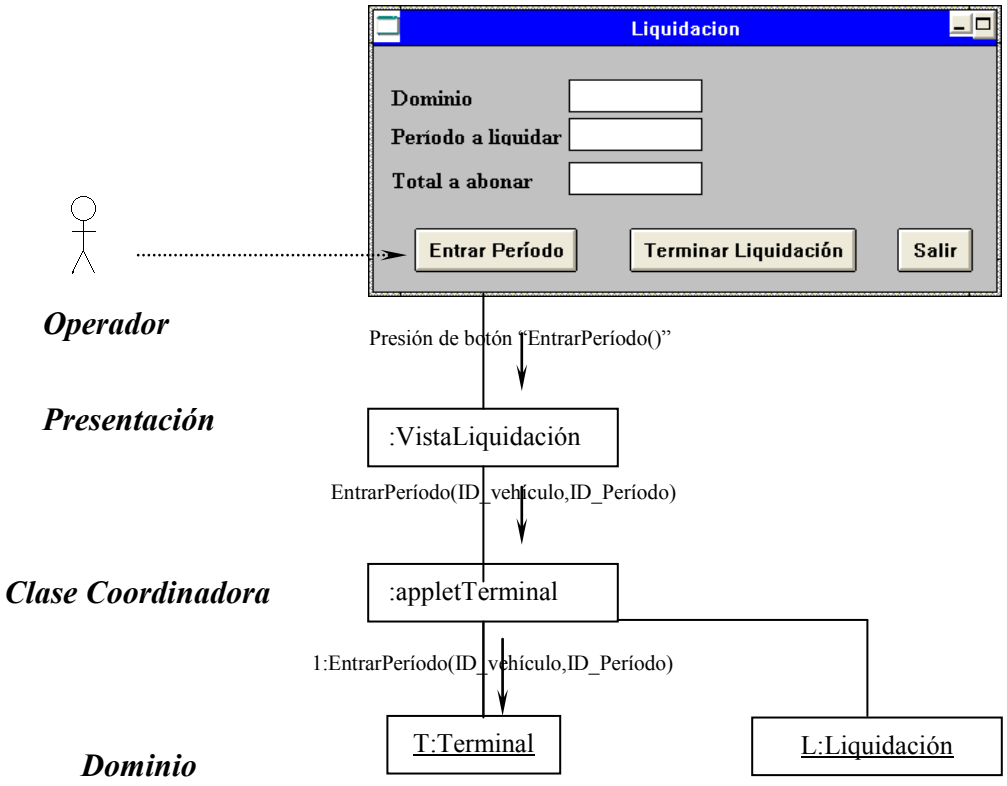


Figura 9.14
Conexión de las clases de dominio y de presentación.

Capítulo 10 - Diagrama de Clases

10.1- Introducción

Un diagrama de clase de diseño ilustra la especificación de las clases de software y sus asociaciones con otras clases. La información típica que él incluye es:

- *clases, asociaciones y atributos*
- *interfases, con sus operaciones y constantes*
- *métodos*
- *información de tipos de atributos*
- *navegabilidad*
- *dependencias*

La definición del diagrama de clases de diseño depende de la creación de los siguientes diagramas:

- *Diagrama de interacción: de éste diagrama se identifica las clases de software que participan en la solución y los métodos de clases.*
- *Modelo conceptual: de éste diagrama se suman los detalles para la definición de clases (atributos)*

*En el modelo conceptual se muestran los **conceptos del mundo real**. En cambio, el diagrama de clases muestra la definición de **entidades de software**.*

10.2- Acerca de la organización del capítulo

En este capítulo realizaremos el diagrama de clase de diseño de la aplicación en tratamiento. Para ello seguiremos los siguientes pasos:

- *Identificaremos y dibujaremos todas las clases participantes en la solución de software (analizando el diagrama de interacción).*

- *Incorporaremos los atributos de cada clase (analizando el modelo conceptual).*

- *Incorporaremos los nombres de los métodos a cada clase (analizando el diagrama de interacción).*

- *Sumaremos la información de tipos a los atributos y métodos.*

- *Sumaremos al dibujo las asociaciones necesarias para soportar la visibilidad de atributo requerida (las asociaciones estarán acompañadas de la flecha de navegabilidad para indicar la dirección de la visibilidad de atributo).*

- *Sumaremos líneas de relación de dependencia para indicar visibilidad no-atributo.*

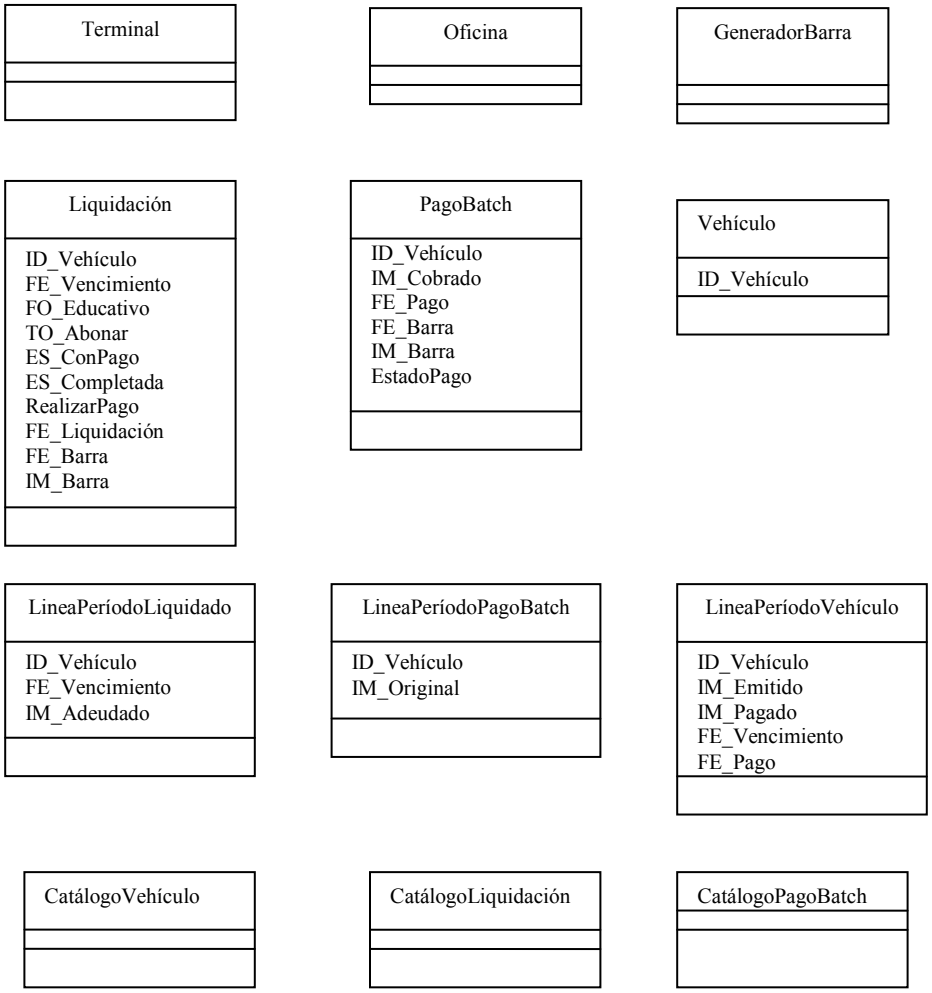
10.3- Identificación de las clases de Software

Mirando todos los diagramas de colaboración vemos que las clases son:

Terminal	*Oficina*	*GeneradorBarra*
Liquidación	*Vehículo*	*PagoBatch*
CatálogoVehículo	*CatálogoLiquidación*	*CatálogoPagoBatch*
LíneaPeríodoLiquidado	*LíneaPeríodoVehículo*	*PeríodoPagoBatch*

10.4- Sumando Atributos a las clases

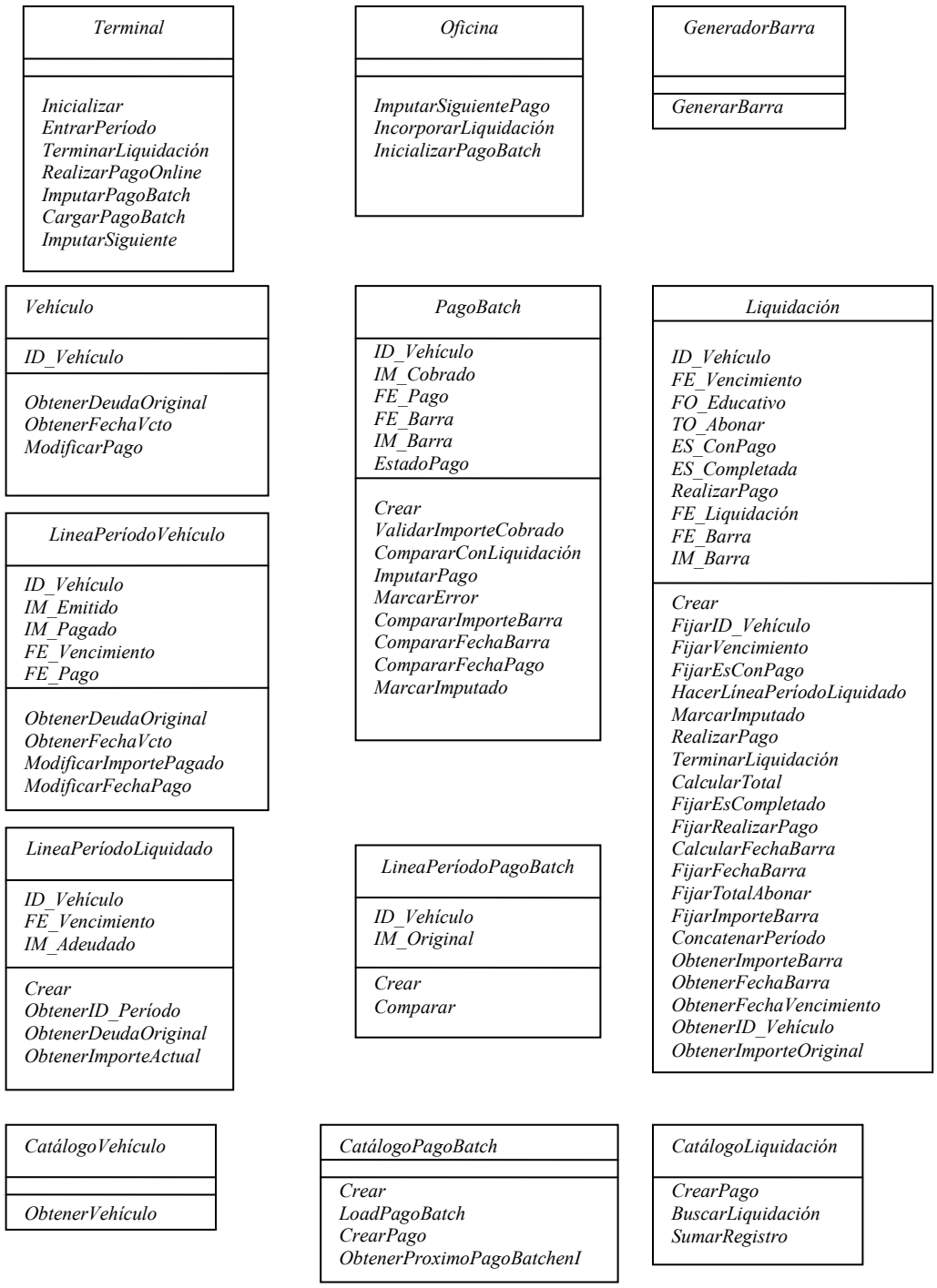
Mirando el modelo conceptual, obtenemos los siguientes atributos:



10.5- Sumando Métodos a las clases

En general, el conjunto de todos los mensajes enviados a la clase X a través de todos los diagramas de colaboración indican la mayoría de los métodos que la clase X debe definirse.

Examinando todos los diagramas de colaboración, los métodos encontrados son:

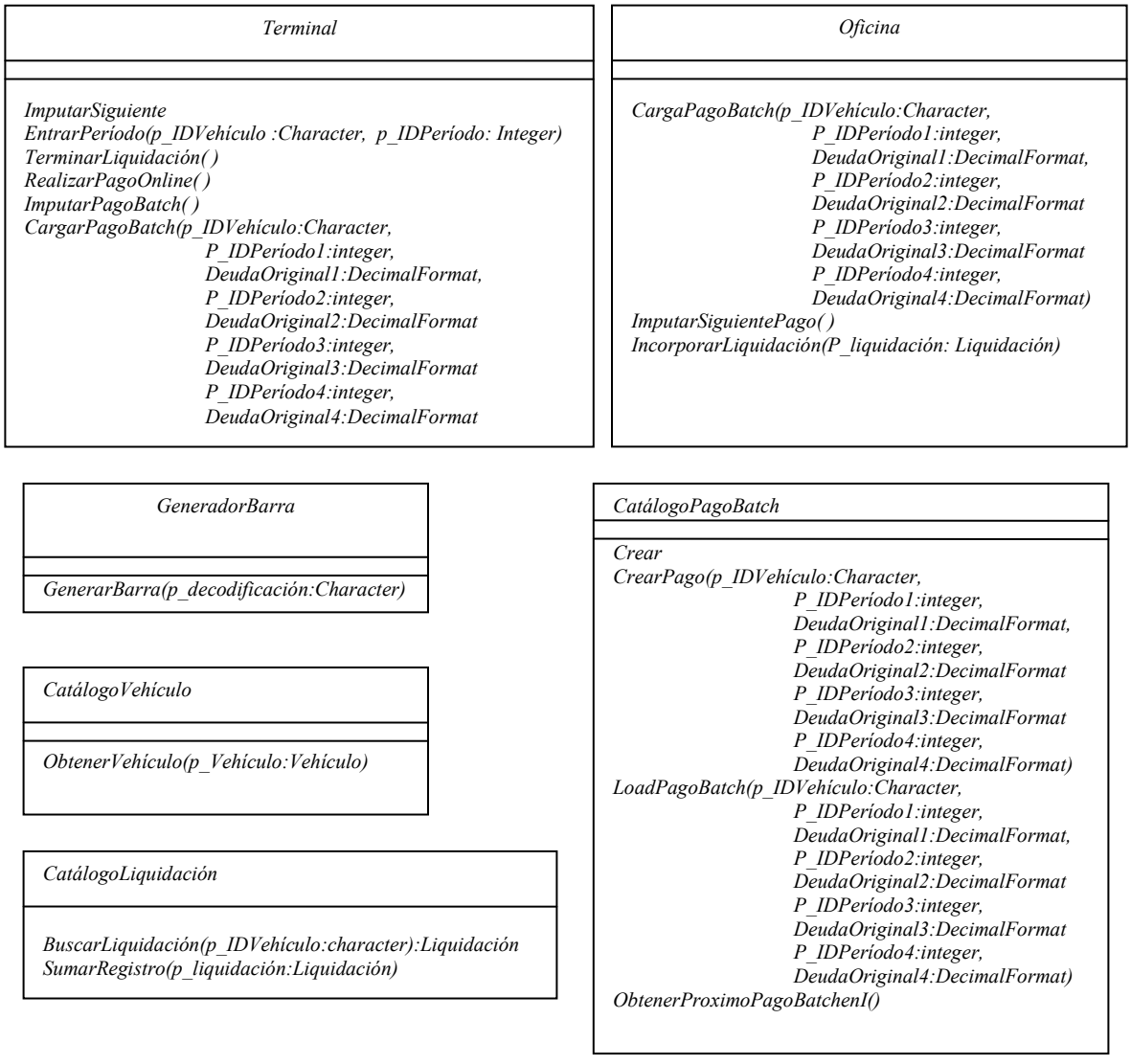


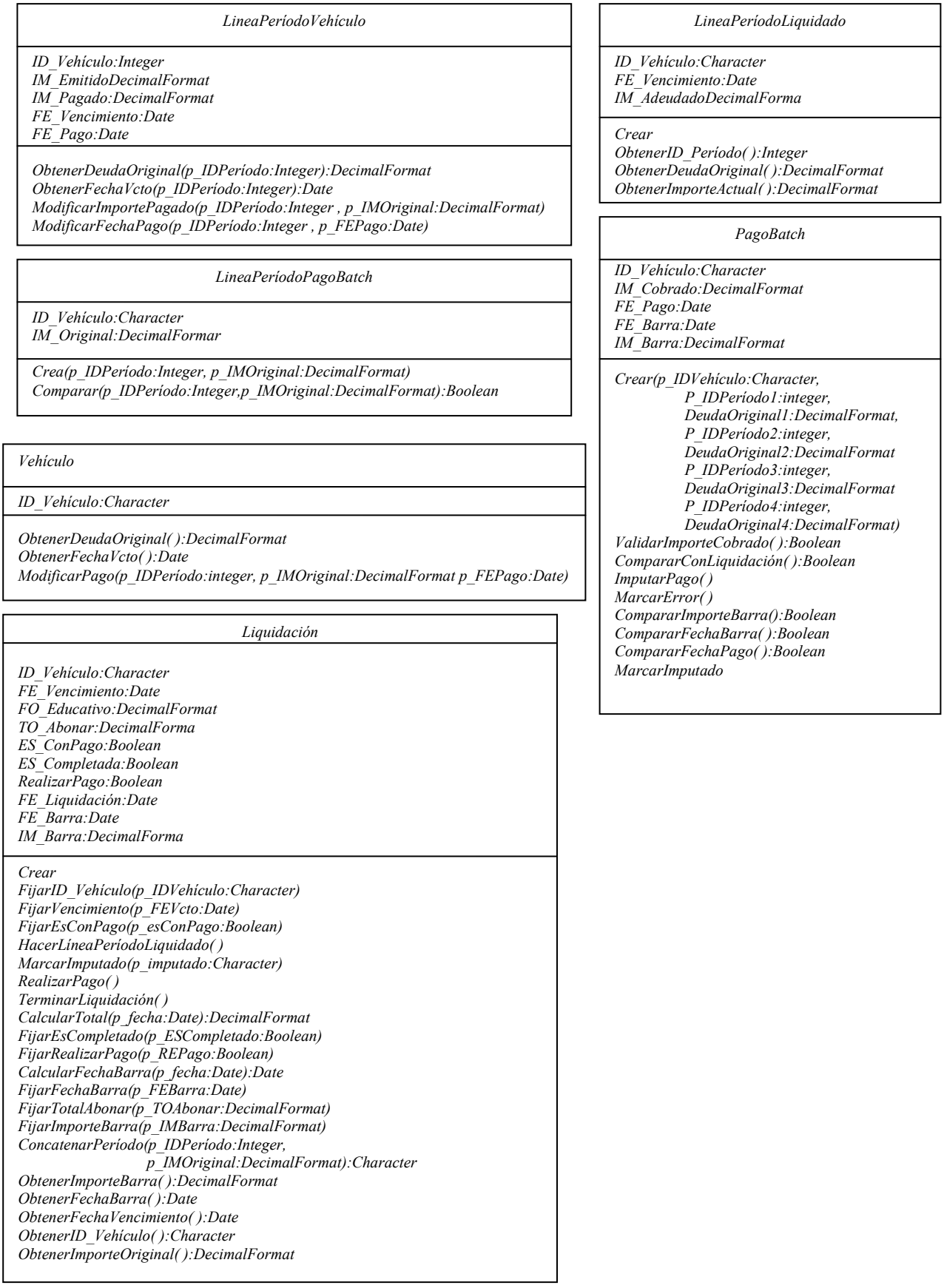
10.6- Sumando tipos a los atributos y métodos

El tipo de los atributos, los parámetros de los métodos y los valores de retorno de los métodos pueden ser mostrados opcionalmente. La decisión de mostrar o no esta información depende de:

- *Si es creado para ser leído por desarrolladores de software entonces los detalles exhaustivos pueden volverse nocivos para la comprensión del diagrama.*
- *Si es creado en una herramienta CASE con generador de código automático entonces los detalles exhaustivos y completos son necesarios.*

Por ejemplo, si mostráramos en forma exhaustiva




LineaPeríodoVehículo
ID_Vehículo:Integer
IM_EmitidoDecimalFormat
IM_Pagado:DecimalFormat
FE_Vencimiento:Date
FE_Pago:Date
ObtenerDeudaOriginal(p_IDPeríodo:Integer):DecimalFormat
ObtenerFechaVcto(p_IDPeríodo:Integer):Date
ModificarImportePagado(p_IDPeríodo:Integer , p_IMOriginal:DecimalFormat)
ModificarFechaPago(p_IDPeríodo:Integer , p_FEPago:Date)
LineaPeríodoPagoBatch
ID_Vehículo:Character
IM_Original:DecimalFormar
Crea(p_IDPeríodo:Integer, p_IMOriginal:DecimalFormat)
Comparar(p_IDPeríodo:Integer,p_IMOriginal:DecimalFormat):Boolean
Vehículo
ID_Vehículo:Character
ObtenerDeudaOriginal( ):DecimalFormat
ObtenerFechaVcto( ):Date
ModificarPago(p_IDPeríodo:integer, p_IMOriginal:DecimalFormat p_FEPago:Date)
Liquidación
ID_Vehículo:Character
FE_Vencimiento:Date
FO_Educativo:DecimalFormat
TO_Abonar:DecimalForma
ES_ConPago:Boolean
ES_Completada:Boolean
RealizarPago:Boolean
FE_Liquidación:Date
FE_Barra:Date
IM_Barra:DecimalForma
Crear
FijarID_Vehículo(p_IDVehículo:Character)
FijarVencimiento(p_FEVcto:Date)
FijarEsConPago(p_esConPago:Boolean)
HacerLíneaPeríodoLiquidado( )
MarcarImputado(p_imputado:Character)
RealizarPago( )
TerminarLiquidación( )
CalcularTotal(p_fecha:Date):DecimalFormat
FijarEsCompletado(p_ESCompletado:Boolean)
FijarRealizarPago(p_REPago:Boolean)
CalcularFechaBarra(p_fecha:Date):Date
FijarFechaBarra(p_FEBarra:Date)
FijarTotalAbonar(p_TOAbonar:DecimalFormat)
FijarImporteBarra(p_IMBarra:DecimalFormat)
ConcatenarPeríodo(p_IDPeríodo:Integer,
p_IMOriginal:DecimalFormat):Character
ObtenerImporteBarra( ):DecimalFormat
ObtenerFechaBarra( ):Date
ObtenerFechaVencimiento( ):Date
ObtenerID_Vehículo( ):Character
ObtenerImporteOriginal( ):DecimalFormat
LineaPeríodoLiquidado
ID_Vehículo:Character
FE_Vencimiento:Date
IM_AdeudadoDecimalForma
Crear
ObtenerID_Período( ):Integer
ObtenerDeudaOriginal( ):DecimalFormat
ObtenerImporteActual( ):DecimalFormat
PagoBatch
ID_Vehículo:Character
IM_Cobrado:DecimalFormat
FE_Pago:Date
FE_Barra:Date
IM_Barra:DecimalFormat
Crear(p_IDVehículo:Character,
P_IDPeríodo1:integer,
DeudaOriginal1:DecimalFormat,
P_IDPeríodo2:integer,
DeudaOriginal2:DecimalFormat
P_IDPeríodo3:integer,
DeudaOriginal3:DecimalFormat
P_IDPeríodo4:integer,
DeudaOriginal4:DecimalFormat)
ValidarImporteCobrado( ):Boolean
ComparарConLiquidación( ):Boolean
ImputarPago( )
MarcarError( )
CompararImporteBarra():Boolean
CompararFechaBarra( ):Boolean
CompararFechaPago( ):Boolean
MarcarImputado

Nótese que si incorporamos toda esta información en el diagrama de clase, el diagrama de clase perderá claridad. Esto nos lleva a optar por una opción intermedia: solo nombraremos los métodos y especificaremos el nombre de los atributos y sus tipos.

10.7- Sumando Asociaciones y navegabilidad

La navegabilidad es una propiedad que indica que es posible navegar uni-direccionalmente a través de una asociación desde los objetos de la fuente a los objetos de destino. La interpretación de una asociación con flecha de navegabilidad es la visibilidad por atributos (ver Anexo Visibilidad) desde la clase fuente a la clase destino. Durante la implementación es usual encontrar que la clase fuente posea un atributo que haga referencia a la clase destino.

El criterio de elección de asociaciones en el diagrama de clases de diseño es: ***Necesita - Ser – Conocido.*** *Este criterio es diferente al del modelo conceptual en el que se puede justificar una asociación con el hecho de que mejora la compresión del dominio del problema.*

Las asociaciones y visibilidades son indicadas por el diagrama de iteración. Examinando cada uno de los diagramas de colaboración realizados en capítulos anteriores, llegamos al diagrama de clase de la figura 10.1.

10.8- Sumando Relaciones de dependencia

La relación de dependencia indica que un elemento (clase, caso de uso, etc.) tiene conocimiento de otro elemento. Se ilustra con una línea discontinua. En el diagrama de clase la relación de dependencia es usada para indicar visibilidad por parámetro, global o declarada localmente (Ver anexo Visibilidad).

En nuestro caso tendremos que cuando creamos una nueva Liquidación la misma es pasada por parámetro en el mensaje de creación a la clase Oficina, luego Oficina tiene una relación de dependencia con Liquidación.

Examinando nuestros diagramas de colaboración obtenemos diagrama de clase de la figura 10.1.

10.9- Diagrama de clase de la aplicación.

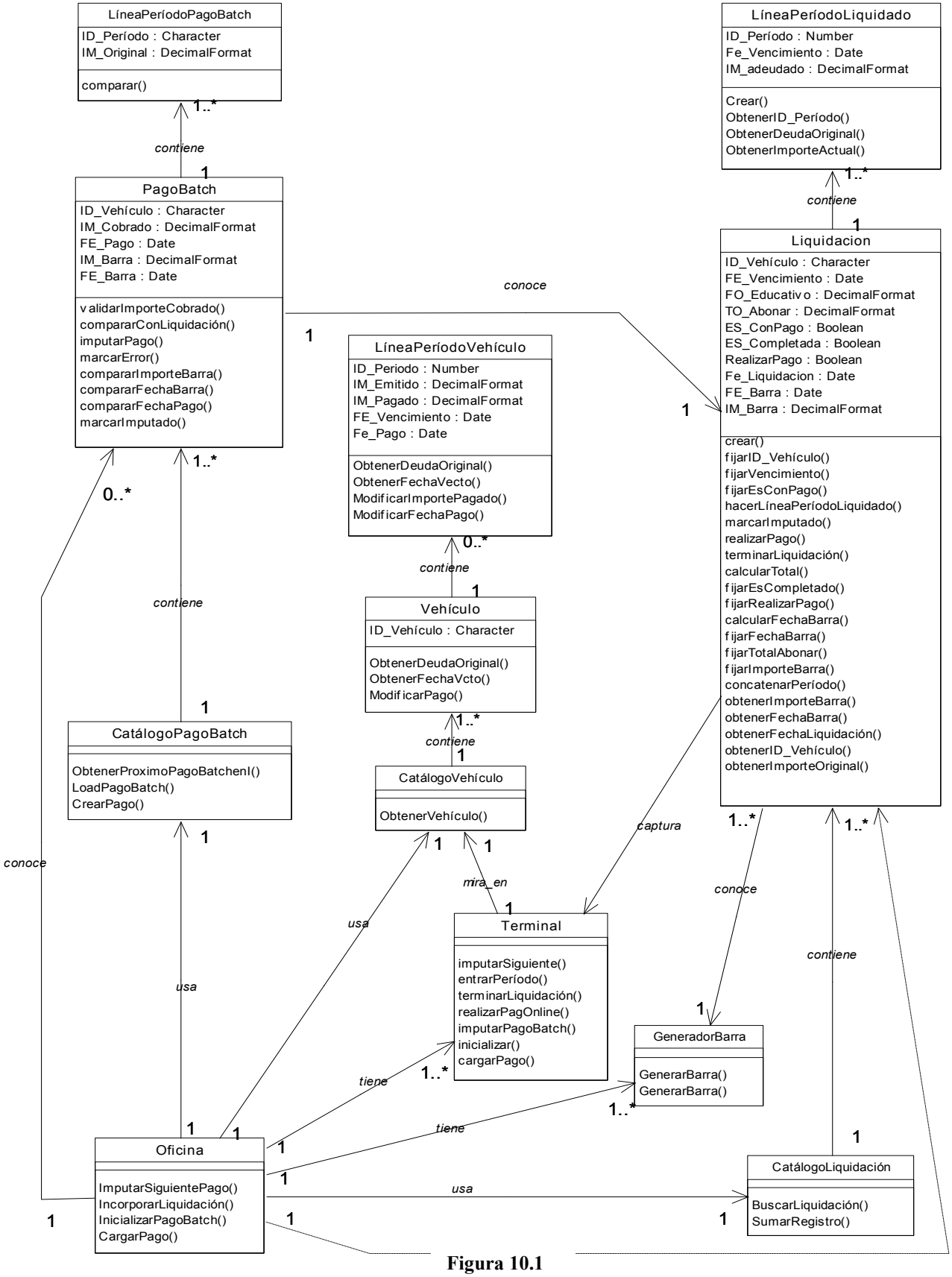


Figura 10.1

Capítulo 11- Descripción del Ciclo de Desarrollo

11.1- Funciones a desarrollar

Recordemos que la idea del proceso unificado de desarrollo de software es la de dividir la aplicación en miniproyectos o ciclos. Cada ciclo resuelve cierta cantidad de funciones y recién después se pasará al siguiente, el cual resuelve y suma nuevas funciones al sistema. Así realizaremos la cantidad necesaria de ciclos o miniproyectos hasta lograr la funcionalidad total de la aplicación. Esta división en miniproyectos hace que el desarrollador no tenga que lidiar con la complejidad de una aplicación completa desarrollándola en su totalidad de una sola vez (vamos desarrollando y probando el sistema en porciones). Además ayuda a su compresión del sistema en forma paulatina (no examinamos la aplicación de golpe, sino en partes manejables) y tiene la ventaja de que una vez terminado el ciclo, el mismo puede ser mostrado al usuario para ver si es lo que él desea o no. En el peor de los casos (de que no satisfaga los requerimientos del cliente), no se habrá perdido el desarrollo de toda la aplicación, sino el desarrollo de un ciclo.

En nuestra aplicación ejemplo, el primer ciclo se caracterizó por la simplificación del problema a tratar para evitar que se complejizara en demasía. Así no tratamos la forma en que se realiza el pago en la oficina de operación y asumimos que en una hoja de liquidación (y en consecuencia en la barra de una liquidación) entraban toda la cantidad de períodos que deseáramos. En el segundo ciclo de vida reveremos el caso de uso Pagar Impuesto, para que soporte el pago con tarjeta de crédito o de banco y además pondremos la restricción de que en una hoja de liquidación o de comprobante de pago puede imprimirse a lo sumo 4 períodos (la cantidad de períodos que entran en una barra).

Las formas de pago con Cheque o al Contado no las consideramos factibles ya que transformaríamos al operador del sistema en un cajero, con toda la infraestructura que esto implica. El pago con tarjeta no trae aparejado el manejo de dinero. Una vez que la imputación es autorizada por en servicio de autorización, es él que se compromete a depositar en la cuenta del Ente Recaudador el dinero correspondiente. Para el operador todo este proceso es transparente, él solo pasa la tarjeta por el post-net, solicita los datos al contribuyente, espera el comprobante de pago y le avisa al contribuyente si es que su pago no ha sido autorizado (en cuyo caso volverá la transacción atrás).

11.2- Asunciones y Simplificaciones de este ciclo

Las siguientes asunciones y simplificaciones serán hechas serán hechas:

- *No se hace mantenimiento de inventario de los pagos (dinero entrante y saliente de la institución) sino que se verá como se asientan los pagos para un contribuyente dado.*
- *El cajero no tiene que hacer login.*
- *No hay control de los pagos realizados por tarjeta de créditos o por Banco.*
- *El operador y la oficina no son mostrados en el recibo.*

- *Solo una forma de pago a la vez es usada (no puede pagar parte por tarjeta de banco y otra por tarjeta de crédito).*
- *Los pagos son hechos en forma completa, no se pueden realizar pagos parciales.*
- *Las tarjetas de crédito y de banco son autorizadas.*
- *La terminal es responsable de comunicarse con el servicio de autorización de créditos que solo envía la información de la tarjeta a la terminal.*
- *La terminal es responsable de comunicarse con el servicio de autorización de banco que solo envía la información de la tarjeta a la terminal.*
- *La comunicación a los servicios externos es vía módem, pero no será tratado en este ciclo.*
- *El servicio de autorización de tarjeta de banco es provisto por el banco.*
- *El servicio de autorización de tarjeta de crédito es provisto por la empresa proveedora de tal servicio.*
- *No importa los períodos que se liquidan en una liquidación pero a lo sumo deben ser 4 por hoja.*
- *Los períodos adeudados son conocidos por el operador.*
- *Los pagos por tarjeta de crédito y por tarjeta de banco son cantidades exactas.*

Capítulo 12- Casos de uso

12.1- Acerca de la organización de este capítulo.

En este capítulo veremos como se modifican los casos de uso para soportar funcionalidades incorporadas en este ciclo.

Por un lado, nuestro sistema debe proveer los mecanismos necesarios para que la cantidad de períodos por hoja impresa no supere a cuatro.

Por otro lado, nuestro sistema debe soportar el pago por tarjeta de crédito y de banco.

En este capítulo examinaremos primero el impacto de la incorporación de la restricción de tener 4 períodos por hoja de impresión. Luego seguiremos con el tratamiento del pago por tarjeta de crédito o de banco.

Aclaración:

Pondremos en letra negrita las modificaciones incorporadas en cada sección.

12.2- Incorporación de restricción de 4 períodos por hoja de impresión.

Examinaremos los casos de uso realizados en el primer ciclo de desarrollo, para estudiar las modificaciones que ellos requieren. Los mismos son:

LiquidarImpuesto
PagarImpuesto
CargarPagoBatch
ImputarPagoBatch

12.2.1- Examinando el caso de uso LiquidarImpuesto.

Caso de uso:	***Liquidar Impuesto***
Actores:	*Contribuyente(iniciador), Operador*
Propósito:	*Realizar una liquidación.*
Descripción:	*El contribuyente arriba a la oficina de operación para solicitar una liquidación para su posterior pago. El operador registra los años cuotas que el contribuyente desea* ***liquidar y le da tantas hojas de liquidación como sean necesarias (teniendo en cuenta que cada hoja posee a lo sumo 4 períodos). El contribuyente se retira con la o las hojas de liquidación en mano****.*
Tipo:	*Primario y esencial*
Referencias cruzadas:	*Funciones: 1.1, 1.2, 1.3, 1.4, 1.6, 1.8.* *Caso de uso: el operador debe tener completado el caso de uso "Log In".*

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el contribuyente arriba a la oficina de operación y da al operador la identificación del vehículo que desea liquidar	
2-El operador marca los años – cuotas que el contribuyente desea se le liquiden.	
	3- Una hoja tendrá a lo sumo 4 períodos. Para cada hoja de liquidación: ***3.1- Determina la deuda actualizada para cada año – cuota de ese vehículo de acuerdo a su fecha de vencimiento, fecha de vencimiento de la liquidación, importe emitido, importe pagado, deuda original, índice de actualización.*** ***3.2- Obtenido dicho valor lo suma al total de la hoja de liquidación*** ***3.3- Calcula los demás datos de la hoja de liquidación: fondo educativo, datos del vehículo, barras de liquidación.***
4-Cuando el operador termina de marcar todos los años cuotas que se liquidarán para ese vehículo, indica al sistema el fin de la selección.	*5- Calcula el **total a pagar** de toda la liquidación* *6- Se imprimen **las hojas de liquidación** de los períodos adeudados.*
*7-El operador le entrega **la/las hojas de liquidación** al contribuyente quien se retira con su liquidación en mano.*	

Cursos alternativos

1 - El código de patente es invalida o inexistente. Se indica con error.

Aclaración:

Notemos que en ese caso de uso ha aparecido la noción de que una liquidación está compuesta de 1 o varias hojas. Cada una de estas hojas las llamaremos hojas de liquidación.

12.2.2- Examinando el caso de uso PagarImpuesto.

Estudiaremos la sección principal de este caso de uso ya que las secciones secundarias no sufren modificación, en este aspecto estamos tratando. Más adelante estudiaremos las secciones secundarias (en el momento de incorporar las formas de pago).

SECCION PRINCIPAL

Caso de uso:	***Pagar Impuesto***
Actores:	*Contribuyente(iniciador), Operador*
Propósito:	*Captura una liquidación y un pago del Impuesto.*
Descripción:	*El contribuyente arriba a la oficina de operación para realizar el pago del Impuesto. El operador registra los años cuotas que el contribuyente desea pagar y luego se asienta el pago, previa autorización del mismo.* ***El contribuyente se retira con el comprobante de pago en mano constituido por 4 períodos por hoja.***
Tipo:	*Primario y esencial*
Referencias cruzadas:	*Funciones: 1.1 a 1.6, 1.8, 1.9, 2.1 a 2.4.* *Caso de uso: el operador debe tener completado el caso de uso "Log In".*

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el contribuyente arriba a la oficina de operación y le informa al operador la patente que desea pagar. *2-El operador marca los años – cuotas que el contribuyente desea pagar.*	***3- Una hoja tendrá a lo sumo 4 períodos. Para cada hoja:*** ***3.1- Determina la deuda actualizada para cada año – cuota de ese vehículo de acuerdo a su fecha de vencimiento, fecha de vencimiento de la liquidación, importe emitido, importe pagado, deuda original, índice de actualización.*** ***3.2- Obtenido dicho valor lo suma al total de la hoja de liquidación*** ***3.3- Calcula los demás datos de la hoja de liquidación: fondo educativo, datos del vehículo.***

4- Cuando el operador termina de marcar todos los años cuotas que desea pagar para ese vehículo, indica al sistema el fin de la selección.	*5- Calcula el **total a pagar** de toda la liquidación.*
6- El operador informa al contribuyente sobre el total a pagar. *7- El contribuyente selecciona el tipo de pago:* - *Si el pago es por tarjeta de crédito, ver sección "Pago por tarjeta de crédito"* - *Si el pago es por tarjeta de cuenta bancaria, ver sección "Pago por Tarjeta de cuenta bancaria"*	*8-Actualización de los datos del vehículo a fin de asentar el importe pagado para cada período marcado para todas las hojas de la liquidación.* *9- Generación de comprobante de pago con a lo sumo 4 períodos por hoja.*
*10-El operador le entrega al contribuyente **el comprobante** de pago.* *11-El contribuyente se retira de la oficina con **el comprobante** de pago en mano.*	

12.2.3- Examinando el caso de uso CargarPagoBatch.

En este caso de uso se cargan los pagos provenientes del banco, dado que no está asociado a las funcionalidades en tratamiento en este ciclo, luego no requiere su cambio.

12.2.4- Examinando el caso de uso ImputarPagoBatch.

Actores:	*Implementador (iniciador)*
Propósito:	*Realizar asentamiento de pagos*
Descripción:	*El implementador dá la orden de ejecución y se produce la imputación o el marcado como erróneo de los pagos que se encuentran en el catálogo de pagos y que están pendientes de imputación. Esta imputación se realizará previa validación.*
Tipo:	*Primario y esencial*
Referencias cruzadas:	*Funciones: 1.5, 1.7, 1.8, 2.5, 2.6*

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el implementador dá la orden de que se proceda a la imputación de los pagos.	*2- Se tomarán todos los pagos cuyo estado de pago indique que se encuentra pendientes de imputación* *3- Se realiza la validación de pago:* *3.1- Reconstruyendo el supuesto cobrado con el coeficiente de actualización (desde la fecha de barra a la fecha de pago) y el importe de la barra y verificando su igualdad con el importe cobrado en los datos del pago.* *3.2- Obteniendo* ***la hoja de liquidación*** *asociada al pago ingresado (se encuentran en el catálogo de liquidaciones). Esta liquidación posee las siguientes características:* - *El importe de barra y fecha de barra es igual al importe y fecha de barra del pago proveniente del banco.* - *La fecha de liquidación es menor o igual a la fecha de pago.* - *Todos los períodos e importes liquidados son iguales a los que contiene el pago proveniente del banco.* *4- Si durante la validación llegamos a la conclusión de que el pago estaba correcto, entonces se imputará:* - *Sumando el importe emitido de cada período ingresado al importe pagado del vehículo para ese período.* - *Modificando el estado del pago para indicar que ya se encuentra imputado.* - *Modificando la liquidación asociada para indicar que ya ingreso el pago.* *5- Si la durante la validación llegamos a la conclusión de que el pago estaba incorrecto, entonces se modificará el estado del pago*

	para indicar que ya se encuentra erróneo. *6-Asienta la información sobre el pago ingresado en el sistema Cuenta de Entrantes (se debe asentar el dinero que el banco le debe a la entidad recaudadora).*
8- El implementador le da los datos de la imputación a la entidad bancaria (Total de toda la imputación, Dominios Imputados.)	*7-Imprime totales sobre los pagos imputados y los marcados como erróneos*

Curso alternativo:

2 - Si no existen pagos pendientes de imputación, entonces se informa de la situación y se termina el proceso.

Aclaración:

El único cambio que sufre este caso de uso es que el pago ya no es comparado con la liquidación sino con una hoja de la misma.

Recordemos que la liquidación está compuesta de 1 a varias hojas. El contribuyente puede pagar las cuotas de una o de todas las hojas liquidadas. De esta manera, el banco reportará uno o varios registros por cada liquidación pagada. Cada registro se corresponderá a cada hoja pagada de la liquidación.

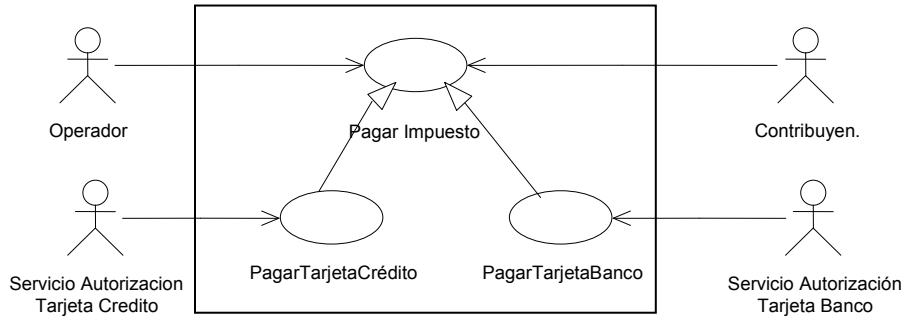
12.3- Incorporación de las formas de pago al sistema.

Las formas de pago son especificadas en el caso de uso PagarImpuesto, luego es este caso de uso que sufrirá modificaciones.

Si bien en la versión del capítulo 3 escribimos las formas de pago como secciones del mismo caso de uso, una alternativa a dicha solución es que sean separados y puestos en otro caso de uso. Hemos decidido separarlas en otros casos de uso, dado que PagarImpuesto es ya suficientemente complejo, y además porque creemos que estas funcionalidades son potencialmente reusables.

12.3.1- Diagrama de caso de uso

Si un caso de uso inicializa o incluye el comportamiento de otro caso de uso se dice que ***usa*** *al segundo caso de uso y a la vez tenemos una relación de uso entre los dos casos de uso. De esta manera, el diagrama de casos de uso se verá de la siguiente manera:*



Relacionando casos de uso con relaciones de uso
Figura 12.1

12.3.2- Caso de uso PagarImpuesto

El curso típico de eventos de PagarImpuesto deberá reflejar la relación de uso de los otros casos de uso.

SECCION PRINCIPAL

Caso de uso:	***Pagar Impuesto***
Actores:	*Contribuyente(iniciador), Operador*
Propósito:	*Captura una liquidación y un pago del Impuesto*
Descripción:	*El contribuyente arriba a la oficina de operación para realizar el pago del Impuesto. El operador registra los años cuotas que el contribuyente desea pagar y luego se asienta el pago, previa autorización del mismo. El contribuyente se retira con el comprobante de pago en mano constituido por 4 períodos por hoja.*
Tipo:	*Primario y esencial.*
Referencias cruzadas:	*Funciones: 1.1 a 1.8.* *Caso de uso: el operador debe tener completado el caso de uso "Log In".*

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el contribuyente arriba a la oficina de operación y le informa al operador la patente que desea pagar	
2-El operador marca los años – cuotas que el contribuyente desea pagar.	*3- Una hoja tendrá a lo sumo 4 períodos. Para cada hoja:* *3.1- Determina la deuda actualizada para cada año – cuota de ese vehículo de acuerdo a su fecha de vencimiento, fecha de vencimiento de la liquidación, importe emitido, importe pagado, deuda original, índice de actualización.* *3.2- Obtenido dicho valor lo suma al total de la liquidación.* *3.3- Calcula los demás datos de la hoja de liquidación: fecha de vencimiento de la liquidación, fondo educativo, datos del vehículo.*
4-Cuando el operador termina de marcar todos los años cuotas que desea pagar para ese vehículo, indica al sistema el fin de la selección.	*5- Calcula el total a pagar de toda la liquidación*
6- El operador informa al contribuyente sobre el total a pagar	

7- El contribuyente selecciona el tipo de pago: - *Si el pago es por tarjeta de crédito* ***inicializar el caso de uso PagarTarjetaCrédito*** - *Si el pago es por tarjeta de banco* ***inicializar el caso de uso PagarTarjetaBanco***	
	8- Actualiza de los datos del vehículo a fin de asentar el importe pagado para cada período marcado para todas las hojas de la liquidación
10-El operador le entrega al contribuyente el comprobante de pago.	*9- Genera el comprobante de pago con 4 períodos por hoja.*
11-El contribuyente se retira de la oficina con el/los comprobante de pago en mano.	

Curso alternativo:

5- El contribuyente no puede pagar y cancela la transacción

6- Si el pago no es aprobado, se solicita otra forma de pago o se cancela la transacción.

Casos de uso relacionados:

- *usa PagarTarjetaCrédito*
- *usa PagarTarjetaBanco*

12.3.3- Caso de uso PagarTarjetaCrédito.

Caso de uso: ***PagarTarjetaCrédito***
Actores: *Contribuyente(iniciador), Operador, Servicio de Autorización. de Tarjeta de Crédito, Cuentas de entrantes.*
Propósito: *Autorizar un pago por tarjeta de Crédito.*

Descripción: *El contribuyente realiza el pago de una liquidación por tarjeta de crédito que es validado por un Servicio de Autorización de Tarjeta de Créditos externo.*

Referencias cruzadas: *Funciones: 1.9, 2.2, 2.4.*

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el contribuyente selecciona la forma de pago por tarjeta de crédito, después que se le informó el total a abonar	

2- El contribuyente comunica la información de la tarjeta de crédito (número de tarjeta, número de DNI)	
3- El operador registra la información de la tarjeta y el requerimiento de autorización de pago por tarjeta de crédito.	*4- Genera el requerimiento de Pago por Tarjeta y lo envía al Servicio Externo de Pago por Tarjeta de Crédito.*
5- El Servicio de Autorización de Tarjeta de Crédito autoriza el pago.	*6- Se recibe la respuesta desde el servicio de Autorización de Pago por Tarjeta de Crédito.*
	7- La terminal le envía la información del pago y de la tarjeta al sistema de cuentas entrantes.
	8- Se muestra el mensaje del suceso de la autorización

Curso Alternativo:

6- La respuesta del Servicio de Autorización de Pago por Tarjeta de Crédito, es denegar el pago entonces se muestra el mensaje del suceso sin realizar el punto 7.

12.3.4- Caso de uso PagarTarjetaBanco.

Caso de uso: ***PagarTarjetaBanco***
Actores: *Contribuyente(iniciador), Operador, Servicio de Autorización, de Cuenta Bancaria, Cuentas de Entrantes*
Propósito: *Autorizar un pago por tarjeta de Banco*

Descripción: *El contribuyente realiza el pago de una liquidación por tarjeta de banco, que es validado por un servicio de Autorización de Tarjeta de Banco externo.*
Referencias cruzadas: *Funciones: 1.9, 2.1, 2.3.*

Curso típico de eventos

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el contribuyente selecciona la forma de pago por tarjeta de banco, después que se le informó el total a abonar.	

2- El contribuyente comunica la información de la tarjeta de banco (número de CBU, número de DNI). *3- El operador registra la información de la tarjeta y el requerimiento de autorización de pago por tarjeta de banco.* *5- El Servicio de Autorización de Tarjeta de Banco autoriza el pago.*	*4- Genera el requerimiento de Pago por Tarjeta de Banco y lo envía al Servicio Externo de Pago por Tarjeta de Banco.* *6- Se recibe la respuesta desde el servicio de Autorización de Pago por Tarjeta de Banco.* *7- La terminal le envía la información del pago y de la tarjeta de banco al sistema de cuentas entrantes.* *8- Se muestra el mensaje del suceso de la autorización*

Curso Alternativo:

6- La respuesta del Servido de Autorización de Pago por Tarjeta de Banco, es denegar el pago entonces se muestra el mensaje del suceso sin realizar el punto 7.

Capítulo 13 - Modelo Conceptual

13.1- Acerca de la organización de este capítulo.

El objetivo de este capítulo es incorporar las nuevas funcionalidades al modelo conceptual obtenido en el primer ciclo.

Comenzaremos analizando la lista de categoría de conceptos.

Luego identificaremos los sustantivos en los cursos típicos de eventos (como lo realizamos en el primer ciclo de desarrollo).

Una vez obtenidos los conceptos candidatos, examinaremos el diagrama conceptual:

- *Lo dividiremos en paquetes para hacerlo más legible y comprensible.*
- *Analizaremos los casos de generalización que aparecen.*
- *Analizaremos los tipos asociativos que aparecen.*
- *Analizaremos las relaciones de agregación existentes.*
- *Juntaremos todos los datos examinados por separado para realizar un nuevo diagrama conceptual de manera de incluirle las funcionalidades en estudio en este ciclo de vida.*

13.2- Encontrando conceptos nuevos de la Lista de Categoría de Conceptos.

En esta lista de conceptos candidatos aparecerán los que consideramos que son nuevos o que sufrirán modificaciones.

Categoría de Conceptos	
Objetos tangibles o físicos	*Tarjeta de Crédito* *Tarjeta Bancaria*
Especificación, diseño o descripción de cosas	
Lugares	
Transacciones	*Pago con Tarjeta de Crédito* *Pago con Tarjeta Bancaria*
Items de la línea de transacción	*Hoja de liquidación*
Roles de personas	
Contenedor de otras cosas	*Liquidación*
Cosas que contiene	*Hojas de Liquidación*
Otros computadores o sistemas mecánicos externos a nuestro sistema	*ServicioAutorizaciónTarjetaCrédito* *ServicioAutorizaciónTarjetaBancaria*
Conceptos no abstractos	*Total a abonar de toda la liquidación* *Total a abonar de la hoja de liquidación*
Organizaciones	
Eventos	
Procesos	
Reglas y pólizas	

Catálogos	
Registros de finanzas, trabajos, contratos y materias legales	*Cuenta de entrantes*
Instrumentos financieros y servicios	
Manuales y libros	

La restricción de que en una hoja de impresión entran a lo sumo 4 períodos, lleva a la modificación del concepto que teníamos de Liquidación:

- *Una liquidación contiene 1 o varias hojas de liquidación, las que a su vez contienen las líneas de los períodos a abonar.*
- *En caso de que el pago se realice en el momento (por alguna de las formas de pago), entonces se abonarán todos los períodos incluidos en la Liquidación; vale decir todos los períodos contenidos en todas las hojas de la liquidación.*
- *Si el pago se realizara vía banco (el contribuyente no abona en el momento de realizar la liquidación, sino que la paga al contado en el banco), se podrá pagar una hoja de toda la liquidación. Así si, por ejemplo, una Liquidación está compuesta 3 hojas, el contribuyente puede optar por pagar 1 de ellas o 2 o todas (tener en cuenta que no se puede pagar parcialmente una Hoja de Liquidación).*

13.3- Encontrando conceptos identificando sustantivos en las frases del curso típico de eventos.

Estudiaremos el curso de eventos de PagarTarjetaBanco y PagarTarjetaCrédito ya que leyendo el curso de eventos de los demás casos de uso modificados vemos que el concepto nuevo que aparecerá es Hoja de Liquidación y que el concepto Liquidación sufrirá cambios.

13.3.1- Identificando sustantivos de curso típico de eventos PagarTarjetaCredito.

Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el contribuyente selecciona la forma de pago por tarjeta de crédito, después que se le informó el total a abonar de la liquidación. *2- El contribuyente comunica la información de la tarjeta de crédito (número de tarjeta, número de DNI).* *3- El operador registra la información de la tarjeta y el requerimiento de autorización de pago por tarjeta de crédito.*	*4- Genera el requerimiento de Pago por Tarjeta y lo envía al Servicio Externo de Pago por Tarjeta de Crédito.*

5- El Servicio de Autorización de Tarjeta de Crédito autoriza el pago.	*6- Se recibe la respuesta desde el servicio de Autorización de Pago por Tarjeta de Crédito.* *7- La terminal le envía la información del pago y de la tarjeta al sistema de cuentas entrantes.* *8- Se muestra el mensaje del suceso de la autorización.*

Los conceptos candidatos que aparecen aquí son:

Contribuyente
Forma de pago por tarjeta de crédito (lo llamaremos PagoTarjetaCrédito)
Total de la liquidación
Información de la tarjeta de crédito:
- *Nro. de Tarjeta,*
- *DNI propietario de tarjeta*

Operador
RequerimientoAutorizaciónTarjetaCrédito
ServicioAutorizaciónTarjetaCrédito
RespuestaServicioAutorizaciónTarjeta
Terminal
Sistema de cuentas entrantes
Tarjeta

Casi todos los conceptos son nuevos salvo el contribuyente, operador, terminal.

El total de la liquidación constituye un atributo de la liquidación ya que representa la suma de cada uno de los totales de cada hoja de la liquidación.

La información de pago serán datos guardados en la cuenta entrantes para posterior reclamo del dinero al ente que corresponda. Dicha información es el total abonado, Nro. de tarjeta, DNI del propietario de la tarjeta, Nro. de liquidación que se pago con esa autorización. Esta operatoria pertenece a otro subsistema de la aplicación que no trataremos en este ciclo de desarrollo.

Los conceptos nuevos que quedan son:

PagoTarjetaCredito
RequerimientoAutorizaciónTarjetaCrédito
ServicioAutorizaciónTarjetaCrédito
RespuestaServicioAutorizaciónTarjeta
Sistema de cuentas entrantes
TarjetaCrédito

13.3.2- Identificando sustantivos de curso típico de eventos PagarTarjetaBanco.

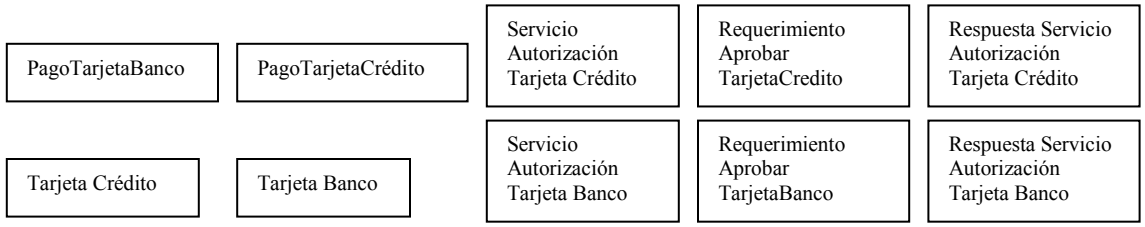
Acción del actor	***Respuesta del sistema***
1-Este caso de uso comienza cuando el contribuyente selecciona la forma de pago por tarjeta de banco, después que se le informó el total a abonar de la liquidación.	
2- El contribuyente comunica la información de la tarjeta de banco(número de CBU, número de DNI).	
3- El operador registra la información de la tarjeta y el requerimiento de autorización de pago por tarjeta de banco.	*4- Genera el requerimiento de Pago por Tarjeta de Banco y lo envía al Servicio Externo de Pago por Tarjeta de Banco.*
5- El Servicio de Autorización de Tarjeta de Banco autoriza el pago.	*6- Se recibe la respuesta desde el servicio de Autorización de Pago por Tarjeta de Banco.*
	7- La terminal le envía la información del pago y de la tarjeta de banco al sistema de cuentas entrantes.
	8- Se muestra el mensaje del suceso de la autorización.

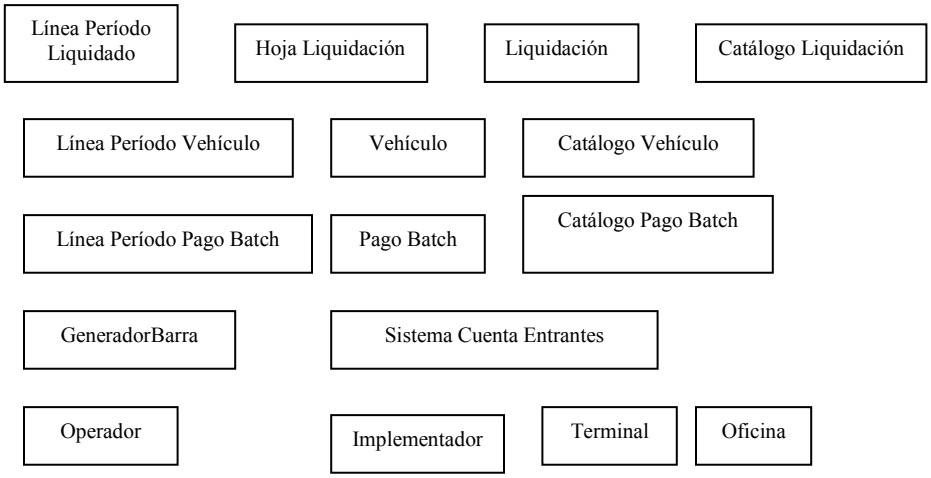
Los conceptos nuevos que quedan son:

PagoTarjetaBanco
RequerimientoAutorizaciónTarjetaBanco
ServicioAutorizaciónTarjetaBanco
RespuestaServicioAutorizaciónTarjetaBanco
Sistema de cuentas entrantes
TarjetaBanco

13.4- Conceptos del diagrama conceptual del segundo ciclo de desarrollo.

Así nuestro modelo conceptual tendrá los siguientes conceptos:





La liquidación tendrá un atributo que será el total de toda la liquidación. El pago por tarjeta de crédito o bancaria tendrán como atributo el número de tarjeta o CBU (en caso de tarjeta de banco) y el DNI del propietario de la misma.

13.5- División del modelo conceptual en paquetes.

Un paquete es un conjunto de elementos del modelo de algún tipo, tal como clases, casos de uso, diagrama de colaboración o otros paquetes.

Los beneficios de la división del modelo conceptual en paquetes es que divide elementos detallados en abstracciones más grandes soportando así que se pueda ver al sistema en una vista de alto nivel y en grupos simples.

Hemos decidido la división del modelo conceptual, puesto que su comprensión y legibilidad en un único diagrama se volverá poco claro, debido al tamaño del mismo (recordemos que han aparecido muchos conceptos nuevos).

Los criterios utilizados para la división en paquetes son, que se deben poner juntos en un mismo paquete aquellos conceptos que:

- *son del mismo área (están relacionados por conceptos o propósitos)*
- *están en una jerarquía de tipos juntos*
- *participan de un mismo caso de uso*
- *están fuertemente asociados*

Para dividir nuestro modelo conceptual en paquetes:

1- *Englobaremos todos los conceptos del modelo conceptual en un paquete que llamaremos Conceptos del Dominio.*

2- Ubicaremos los conceptos más comunes en un paquete que llamaremos Conceptos Comunes.

3- Usando los criterios antes expuestos obtendremos los demás paquetes de la aplicación.

Basándonos en los conceptos expuestos arriba obtenemos la siguiente división.

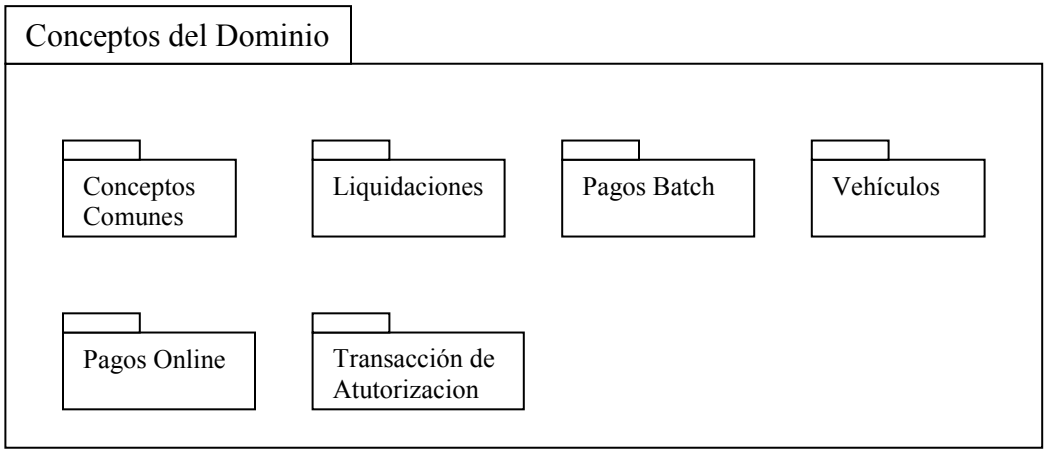


Figura 13.1
Paquetes incluidos en paquete "Conceptos del dominio".

Los conceptos incluidos en cada paquete serán:

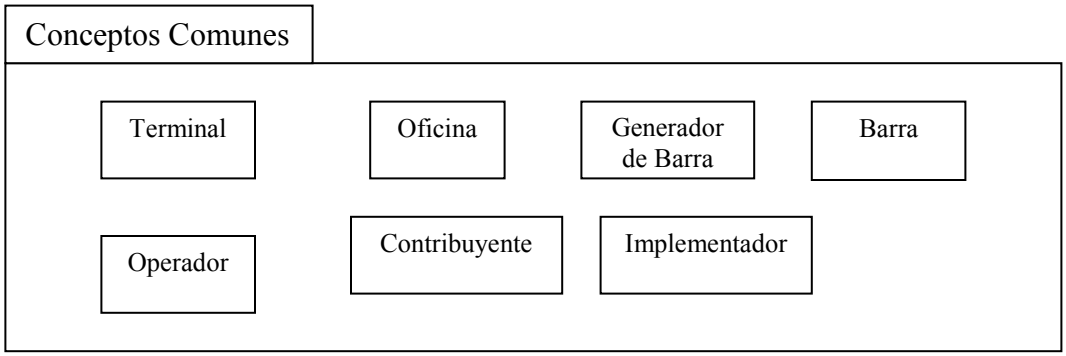


Figura 13.2
Conceptos incluidos en el paquete "Conceptos Comunes".

Tanto terminal y oficina constituyen conceptos comunes a todos los casos de uso.

Por el otro lado consideramos que el Generador de Barras y la Barra pueden ser usadas en cada función donde se requiera la generación de una barra, es por eso que la ponemos en Conceptos comunes y no en Liquidaciones (en este ciclo de vida en el único lugar donde la usamos es durante la Liquidación).

El contribuyente, operador e implementador constituyen actores que utilizan el sistema para obtener varias funciones de él (sin embargo, no solo la liquidación o la imputación de pagos). Es por ella que hemos decidido ponerlo en este paquete.

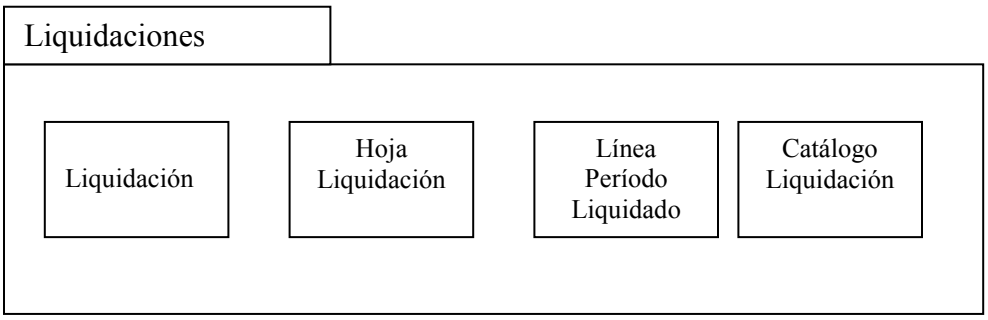


Figura 13.3
Conceptos incluidos en el paquete "Liquidaciones"

Los conceptos de este paquete están asociados por propósito, tanto la Liquidación, la HojaLiquidación, la LíneaPeríodoLiquidado o el CatálogoLiquidación tienen como propósito de alojar los datos de las liquidaciones realizadas.

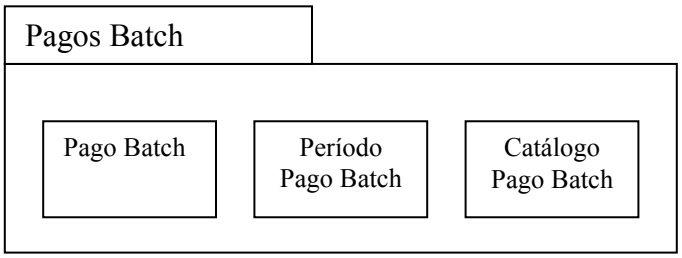


Figura 13.4
Conceptos incluidos en el paquete "Pagos Batch"

Los conceptos de este paquete están asociados por propósito, tanto la PagoBatch, el PeríodoPagoBatch o el CatálogoPagoBatch tienen como propósito alojar los datos de un nuevo pago enviado por el Banco.

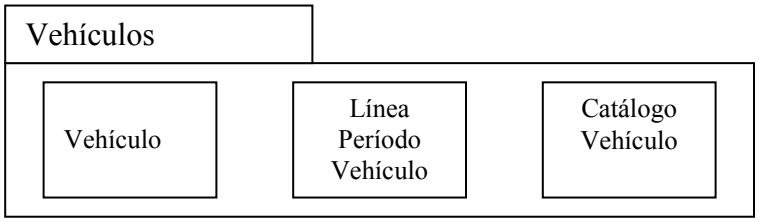


Figura 13.5
Conceptos incluidos en el paquete "Vehículo"

Los conceptos de este paquete están asociados por propósito, tanto la Vehículo, la LíneaPeríodoVehículo o el CatálogoVehículo tienen como propósito alojar los datos de un Vehículo.

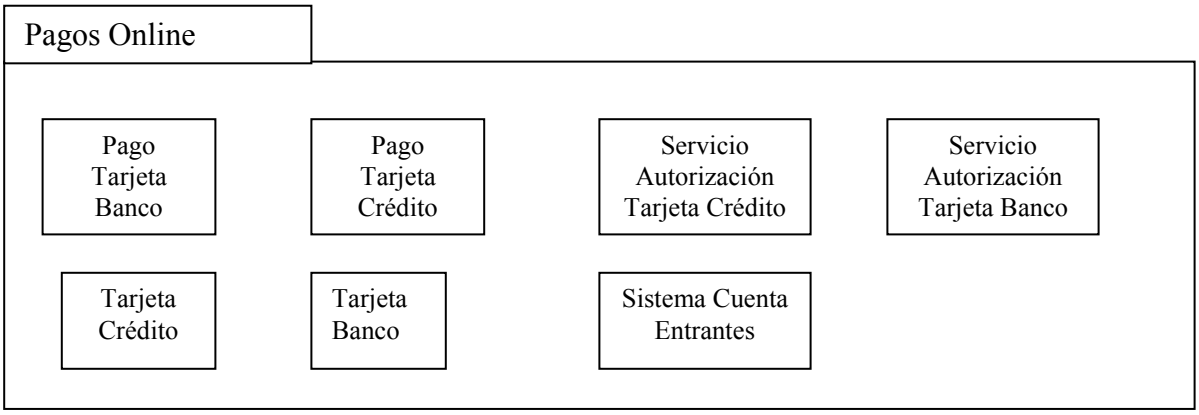


Figura 13.6
Conceptos incluidos en el paquete "Pago Online"

Los conceptos en este paquete están asociados al tratamiento de las formas de pago.

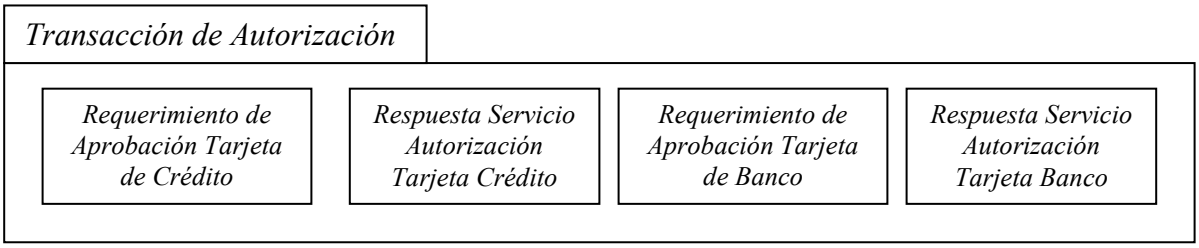


Figura 13.7
Conceptos incluidos en el paquete "Transacción de Autorización"

Los conceptos en este paquete están agrupados por jerarquía de clases.

13.6- Jerarquías de tipos en el Modelo Conceptual

Los conceptos PagoTarjetaCrédito y PagoTarjetaBanco son similares. Es posible agruparlos en una jerarquía de tipos generalización – especialización con una clase PagoOnline como superclase. La clase PagoOnline será una clase abstracta y así el modelo conceptual quedará de la siguiente forma:

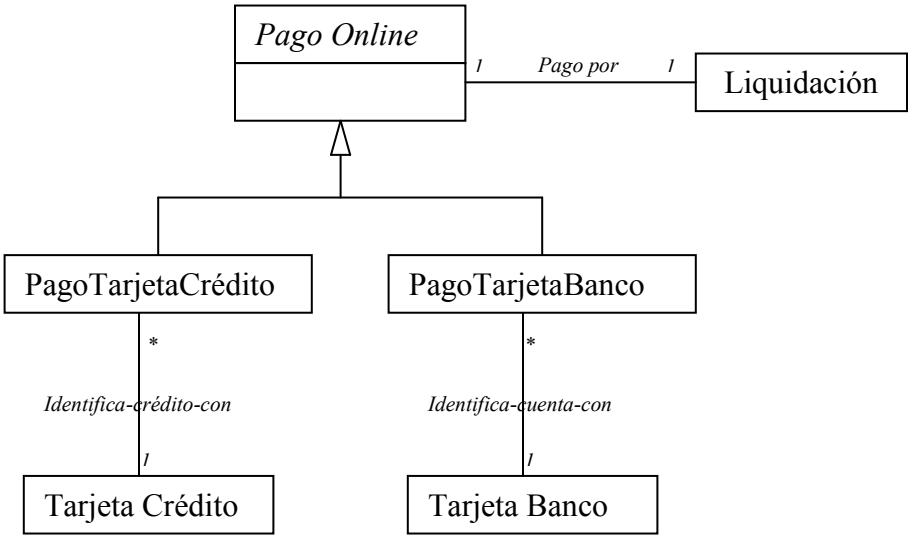


Figura 13.8
Jerarquía de conceptos en "Pago Online"

Por otro lado el servicio de autorización por tarjeta de crédito o tarjeta de banco son variaciones de conceptos similares y tiene atributos comunes de interés.

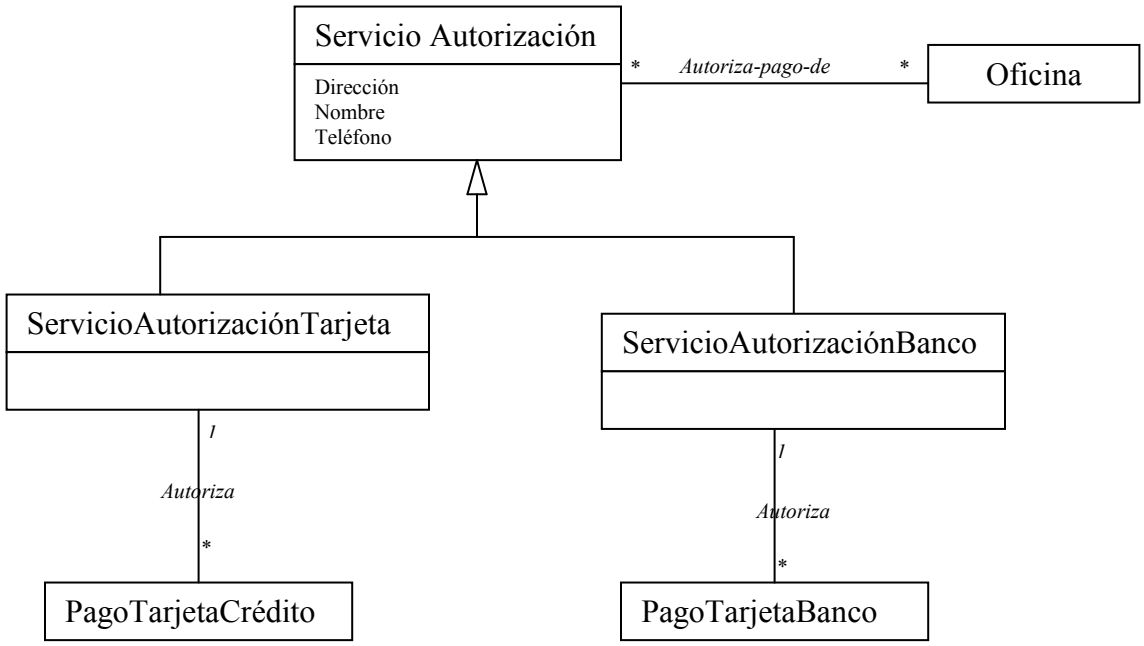


Figura 13.9
Jerarquía de conceptos en "Servicio de Autorización"

Por último tenemos es útil mostrar en el modelo conceptual las transacciones de servicio de actualización (requerimiento y aprobación) dado que los procesos y actividades del pago Online se desarrollan alrededor de ellos.

Una estructura jerárquica posible sería:

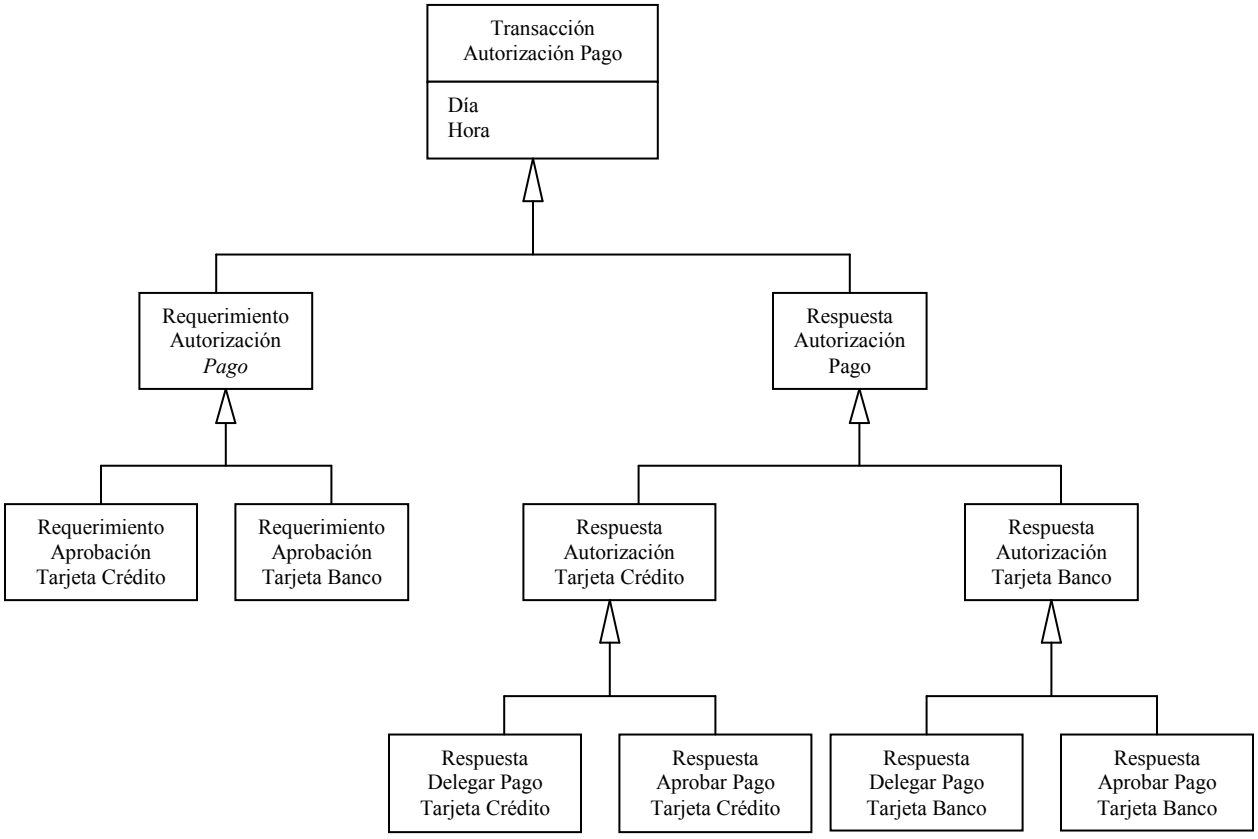


Figura 13.10
Jerarquía de conceptos en la "Transacción de Autorización de Pago"

13.7- Tipos Asociativos que aparecen en el Modelo Conceptual.

En un tipo asociativo podemos sumar características a una asociación, es decir las características no pertenecen a una clase particular sino a la asociación que existe entre 2 clases.

En nuestro modelo conceptual tenemos que:

- *El servicio de autorización requiere asignar un ID de pago a autorizar para la identificación durante la comunicación.*
- *El requerimiento de autorización de pago desde la oficina al servicio de autorización requiere la inclusión del ID de pago que identifica la oficina de rentas al servicio.*

- *La oficina tiene un ID diferente para cada Servicio de autorización.*

La pregunta es ¿Dónde ubicar el atributo ID del pago?

- *Si lo ponemos en la Oficina estaremos cometiendo un error ya que ya que una Oficina puede tener más de un valor de ID de pago (dependiendo del servicio de autorización)*
- *Si lo ponemos en el Servicio de Autorización también estaremos equivocados ya que el Servicio de Autorización posee más de un ID de pago (dependiendo de la oficina de la cual se solicita la autorización)*

Luego llegamos a la conclusión que la estructura adecuada es la de creación de un tipo asociativo quedando el diagrama de la siguiente forma:

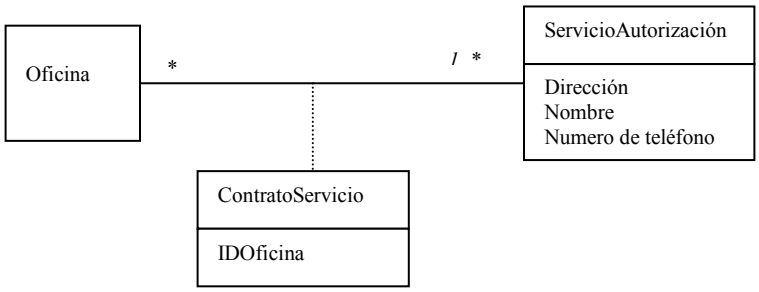


Figura 13.11
Representación de tipo asociativo entre Oficina y Servicio de Autorización

El concepto ContratoServicio esta modelado como tipo asociativo relacionado a la relación entre Oficina y Servicio de Autorización.

Otra asociación con clase asociativa que aparece en el modelo conceptual es la que existe entre el Servicio de autorización de Tarjeta de Banco y Pago por Tarjeta de Banco y la asociación entre el Servicio de Autorización de Tarjeta de Crédito y Pago por Tarjeta de Crédito que tiene como clase asociativa la Respuesta del Servicio de Autorización. El esquema quedará de la siguiente forma:

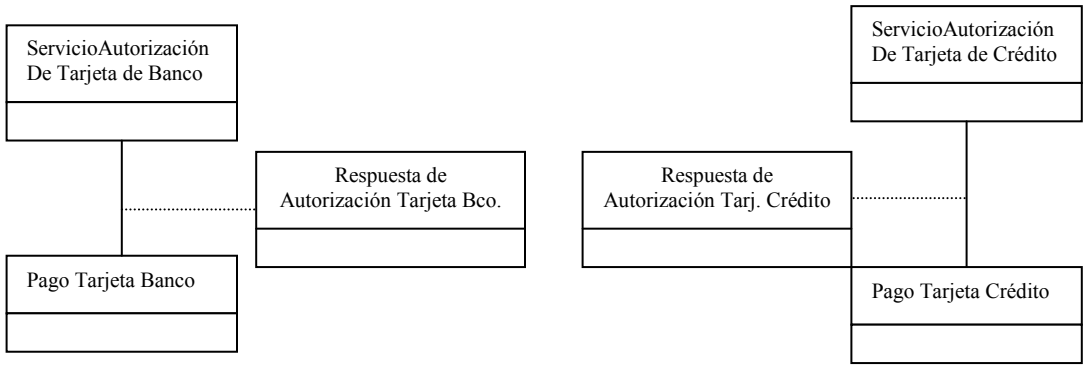


Figura 13.12
Representación de tipos asociativos entre:
1- El Servicio de Autorización de la Tarjeta de Banco y el Pago por Tarjeta de Banco.
2- El Servicio de Autorización de la Tarjeta de Crédito y el Pago por Tarjeta de Crédito.

13.8- Incorporación de las relaciones de agregación al Modelo Conceptual.

Hemos decidido marcar las relaciones de agregación en las asociaciones donde aparecen ya que su trae aparejadas las siguientes ventajas:

- *Clarifica el dominio del problema al identificar la existencia de partes independientes de un todo.*
- *Ayuda a la identificación de una clase creadora (usando el patrón GRASP de Creación[Larman97]).*
- *Las operaciones del todo (tales como creación o borrado) frecuentemente se propagan a las partes.*
- *Identificar el todo respecto a las partes soporta encapsulamiento (se ocultan las partes en el todo).*

13.9- Diagrama del Modelo Conceptual.

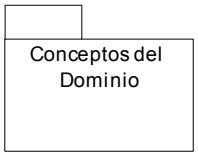


Figura 13.13
Paquete principal del Modelo Conceptual

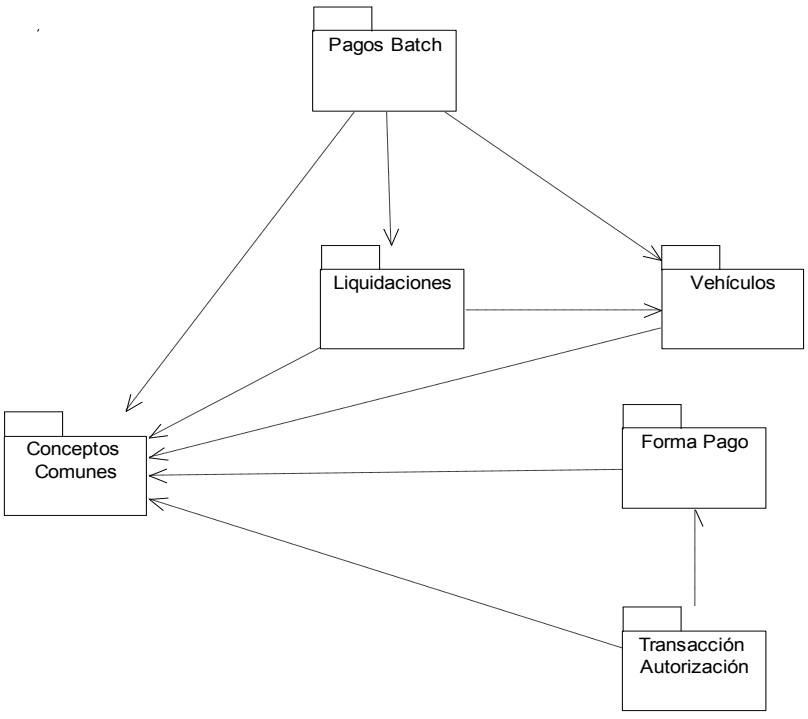


Figura 13.14
Relaciones entre los paquetes del Modelo Conceptual

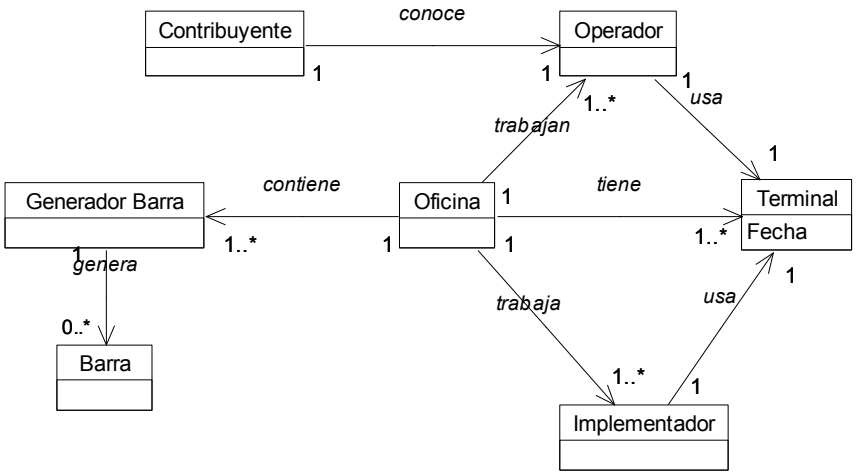


Figura 13.15
Paquete de "Conceptos Comunes"

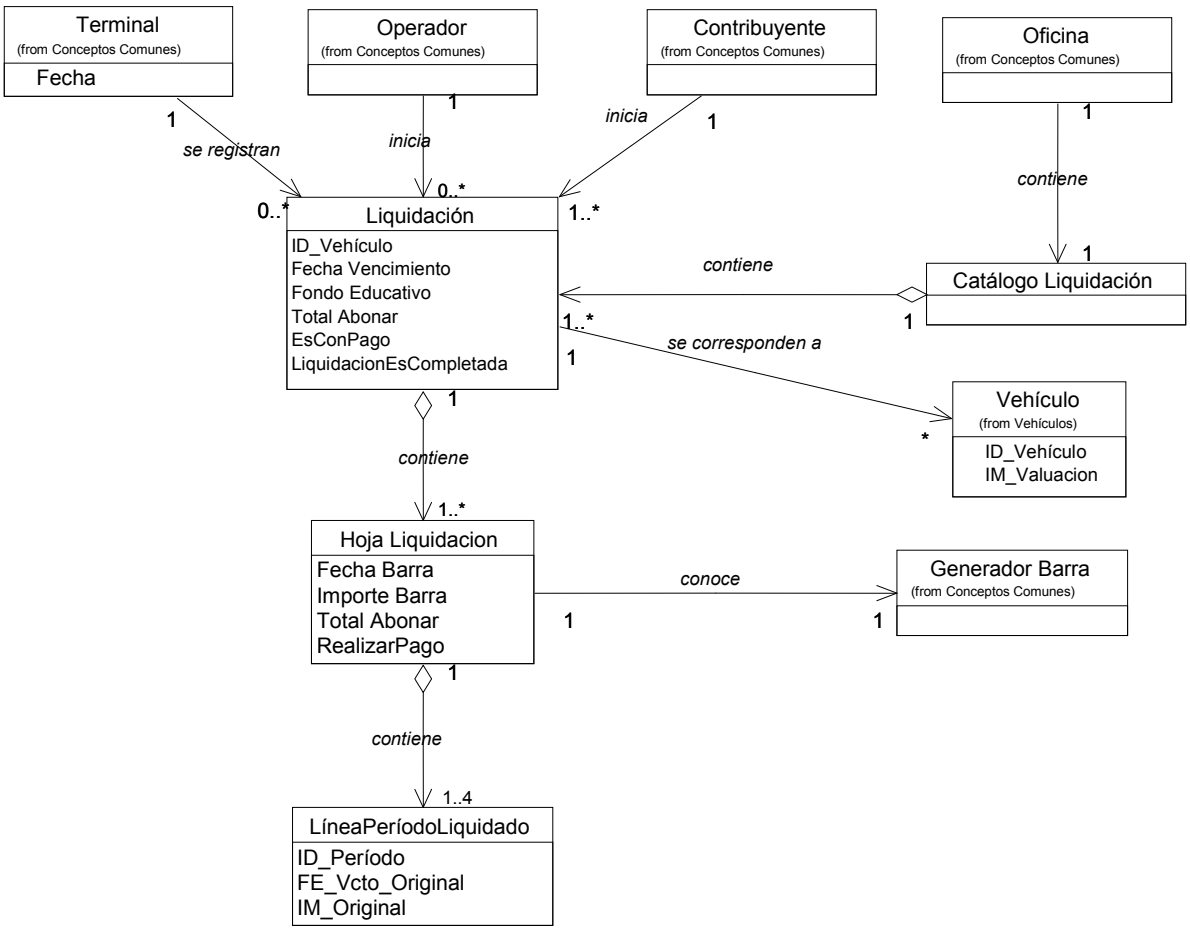


Figura 13.16
Paquete de "Liquidación"

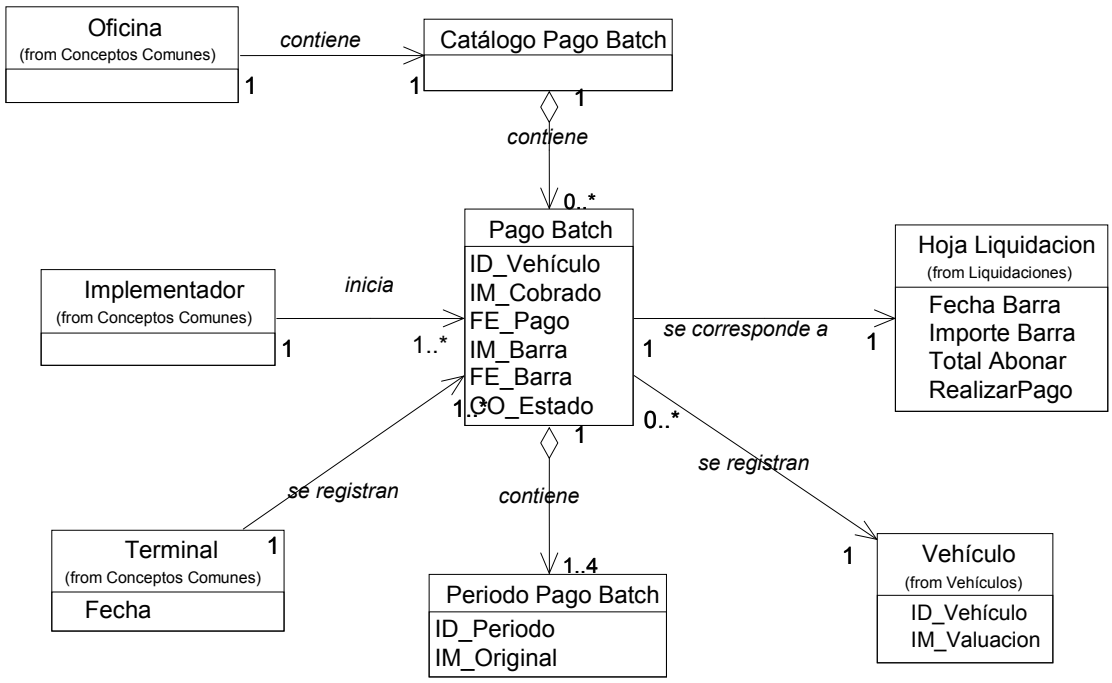


Figura 13.16
Paquete de "PagoBatch"

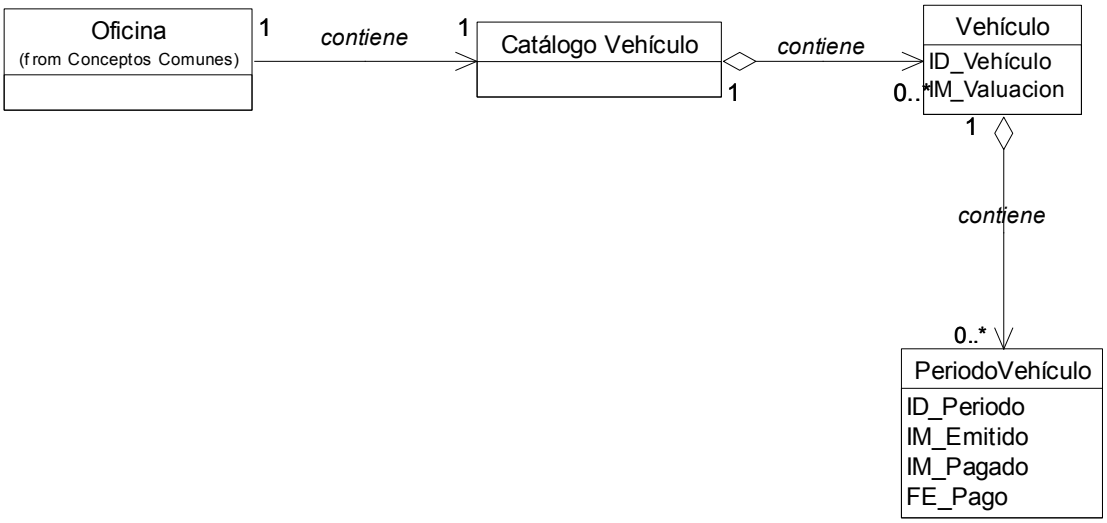


Figura 13.17
Paquete de "Vehículo"

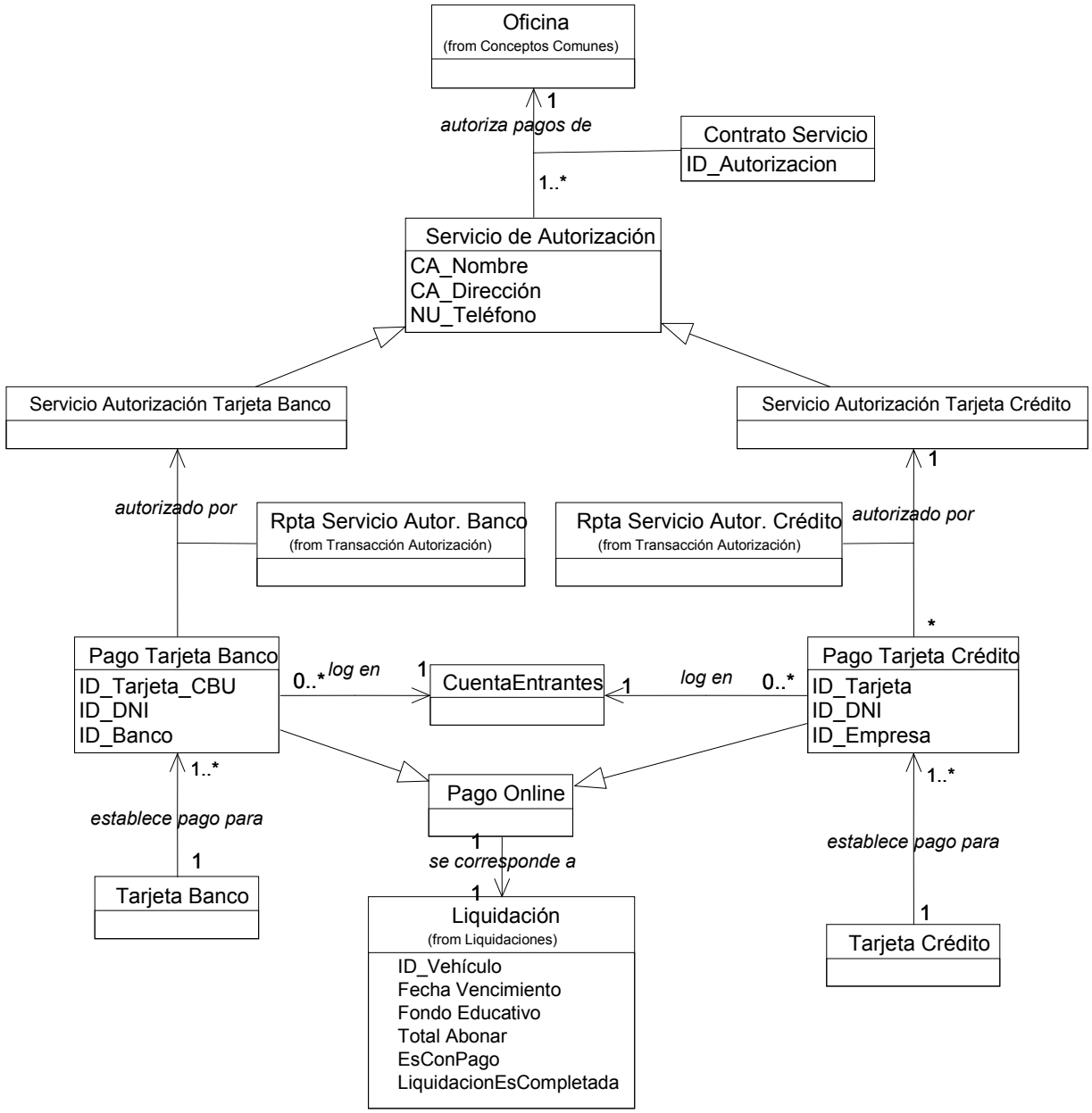


Figura 13.18
Paquete de "Forma de Pago"

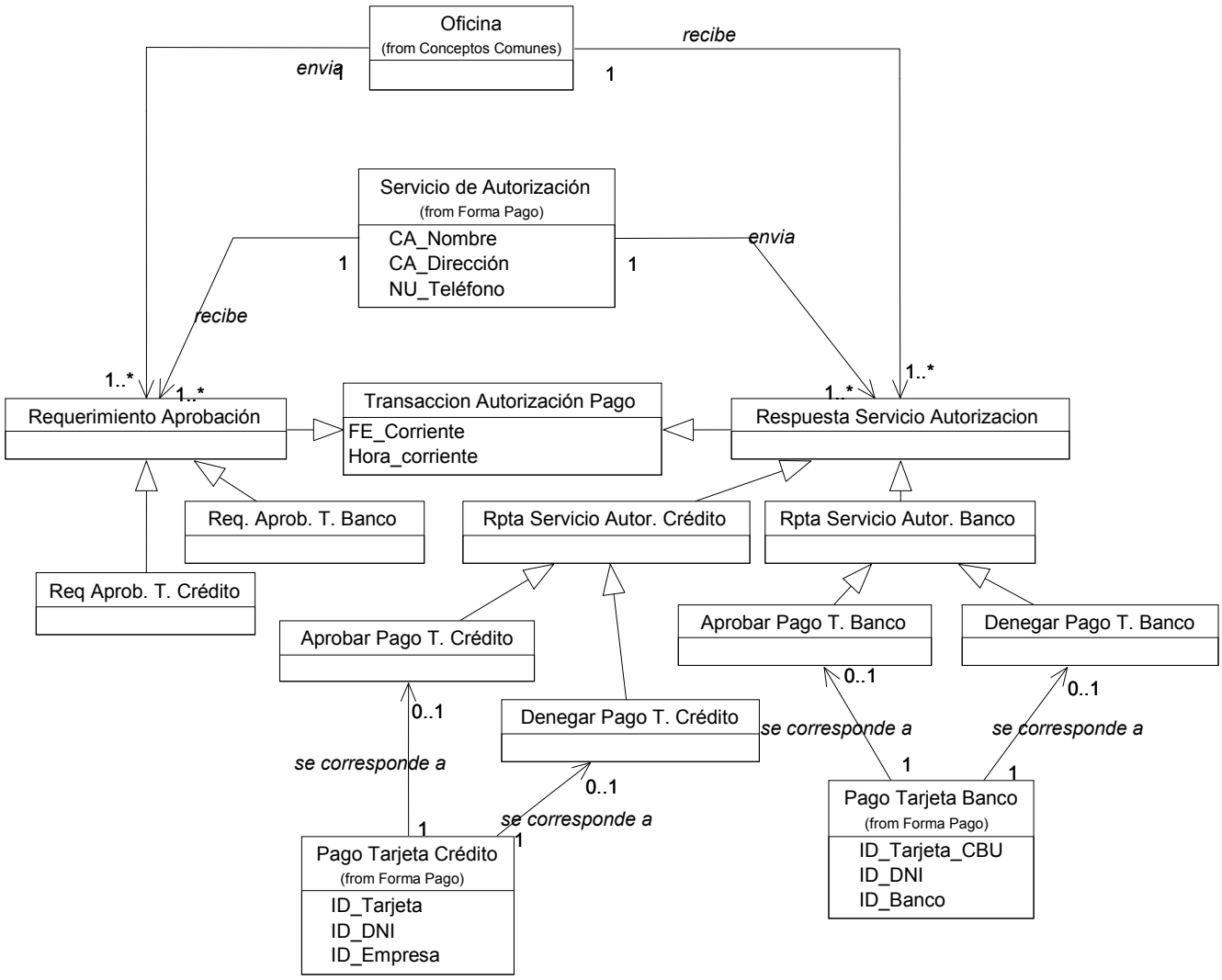


Figura 13.19
Paquete de "Transacciones de Autorización"

Capítulo 14 - Diagramas de secuencia y Contratos

14.1- Acerca de la organización del capítulo.

En este capítulo definiremos primero los diagramas de secuencias del nuevo ciclo en desarrollo.

A continuación revisaremos los contratos de sistema existentes (para incorporar la restricción de los 4 períodos por hoja).

Por último escribiremos los contratos nuevos que surjan del diagrama de secuencias.

14.2- Diagrama de secuencias.

El caso de uso PagarImpuesto es el que sufrió cambios que requieren la revisión del diagrama de secuencias.

14.2.1- Diagrama de secuencias de Pagar Impuesto con Autorización aceptada.

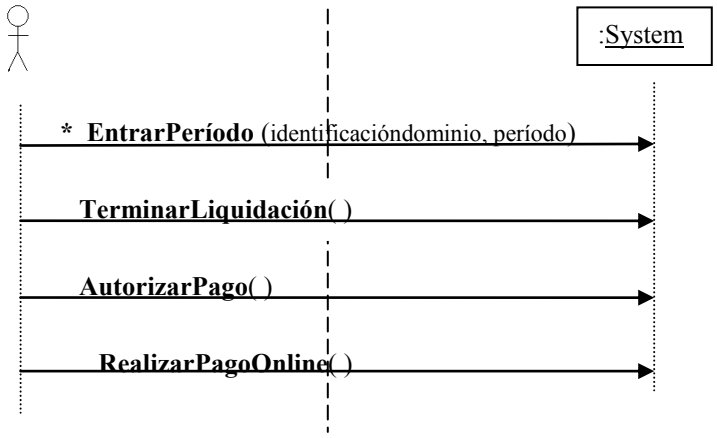


Figura 14.1

14.2.2- Diagrama de secuencias de Pagar Impuesto con Autorización denegado.

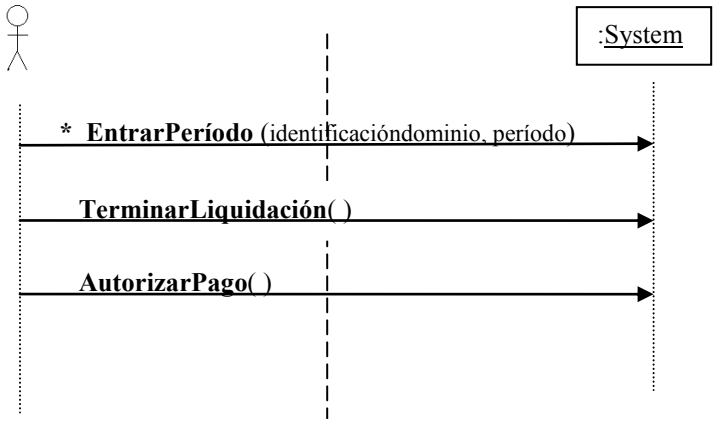


Figura 14.2

14.2.3- Diagrama de secuencias de AutorizarPago:PagoTarjetaCrédito.

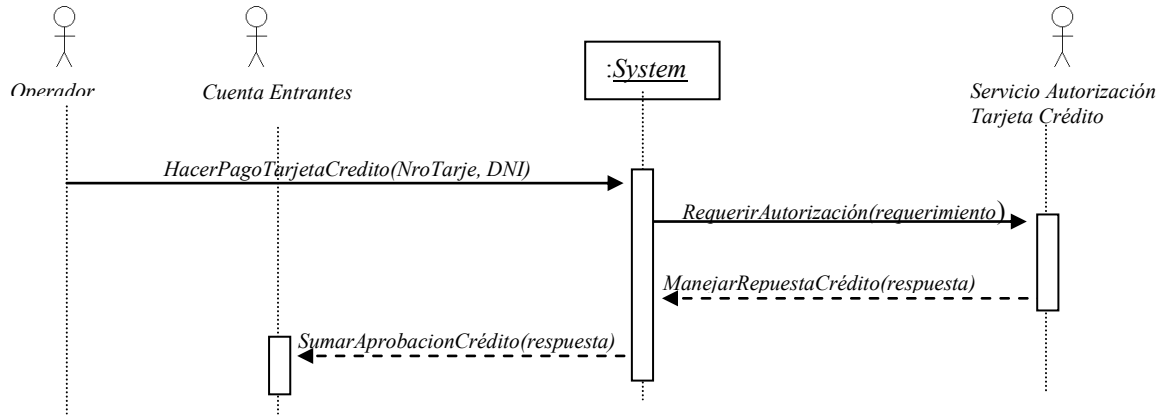


Figura 14.3

14.2.4- Diagrama de secuencias de AutorizarPago:PagoTarjetaBanco.

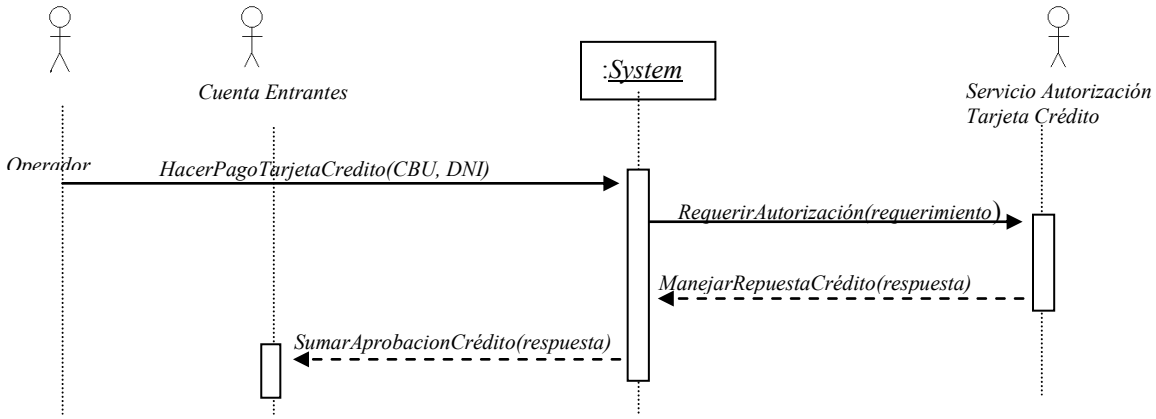


Figura 14.4

14.3- Operaciones del sistema.

En base al diagrama de secuencias surgen las siguiente operaciones de sistema (se marcan con negrita las nuevas):

1- EntrarPeríodo (identificacióndominio, período)
2- TerminarLiquidación( )
3- RealizarPagoOnline( )
4- InicializarPagoBatch
5- CargarPagoBatch(datos del pago)
6- ImputarPagoBatch

7- HacerPagoTarjetaCrédito
8- HacerPagoTarjetaBanco

Aclaración: *En RealizarPagoOnline se realiza el proceso de asentamiento de los pagos para los períodos de un vehículo dado, mientras que HacerPagoTarjetaCrédito o HacerPagoTarjetaBanco representa la operación de autorización del pago por tarjeta de Crédito o por Tarjeta de Banco respectivamente.*

14.4- Contratos del sistema.

En esta sección, desarrollaremos primero los contratos de sistema que han sufrido modificaciones debido a la restricción de 4 períodos por hoja de liquidación. Para indicar la eliminación de post-condiciones, tacharemos aquellas post-condiciones que existían en la versión anterior y resaltaremos aquellas que han sufrido modificaciones.

Luego crearemos los nuevos contratos correspondientes a las operaciones para el pago por tarjeta de crédito y de banco.

Las operaciones de sistema InicializarPagoBatch y CargarPagoBatch, no serán mencionadas en este capítulo ya que no poseen modificaciones.

14.4.1- Revisión del contrato de sistema EntrarPeríodo.

Nombre:	*EntrarPeríodo (identificación del dominio: alfa-numérico período: numérico).*
Responsabilidades:	*Ingresar la liquidación de un período y sumar el importe liquidado al total de la liquidación.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 1.1, 1.2, 1.3, 1.4, 1.6* *Casos de Uso: Liquidar Impuesto* *Pagar Impuesto*
Excepciones:	*Si la identificación del dominio es inválida indica que hay un error.*
Pre-condiciones:	*La identificación del dominio es conocida por el sistema.* *El período existe para el vehículo.* *Las características del vehículo y los períodos emitidos para ese vehículo son conocidos por el sistema.*

Post-condiciones:

- *Si fue una nueva Liquidación entonces:*
  - *La Liquidación fue creada.*
  - *La nueva Liquidación fue asociada a la terminal.*
  - *La nueva Liquidación fue asociada a un Vehículo, basado en la identificación del vehículo.*

- *La Liquidación.ID_Vehículo fue fijado.*
- *La Liquidación.esConPago fue fijado.*
- *La Liquidación.VencimientoLiquidación fue fijado.*
- *La Liquidación.FechaLiquidación fue fijado.*
- *La Liquidación.FondoEducativo fue fijado.*

- ***Si es una nueva liquidación o la cantidad de períodos en la hoja es 4 entonces***
  - ***Una hoja de Liquidación fue creada.***
  - ***La Hoja de Liquidación fue asociada a la liquidación.***
- ***Si la cantidad de períodos en la hoja es de 4***
  - ***La HojadeLiquidacion.TotalAbonar fue calculado.***
- ***Si la cantidad de períodos en la hoja es de 4 y la liquidación.esConPago se encuentra en false entonces:***
  - ***La HojaLiquidación.FechadeBarra fue fijada.***
  - ***La HojaLiquidación.Importedebarra fue calculada.***
  - ***La HojaLiquidación fue asociada al Generador de Barra.***
  - ***Las 2 barras fueron creadas***
  - ***Generador de Barra fue asociada a la o las barras.***
- *Una LíneaPeríodoLiquidado fue creada.*
- *La* ***Hoja de Liquidación*** *fue asociada a la nueva LíneaPeríodoLiquidado.*
- *La LíneaPeríodoLiquidado.PeríodoLiquidado fue fijado.*
- *La LíneaPeríodoLiquidado.VencimientoOriginal fue fijado.*
- *La LíneaPeríodoLiquidado.DeudaOriginal fue fijado.*

14.4.2- Revisión del contrato de sistema TerminarLiquidación.

Nombre:	*TerminarLiquidación*
Responsabilidades:	*Registrar el fin de la liquidación, mostrar y asentar los datos de la liquidación.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 1.1, 1.2, 1.6, 1.8* *Casos de Uso: Liquidar Impuesto* *Pagar Impuesto*
Excepciones:	*Si no se marca al menos un período indica un error.*
Pre-condiciones:	*La identificación del dominio es conocida por el sistema* *Los períodos a liquidar son conocidos por el sistema* *Las características del vehículo y los períodos emitidos para ese vehículo son conocidos por el sistema.*

Post-condiciones:

- *Liquidación.esCompletada fue fijada en true (modificación de atributo) (nuevo atributo)*
- ***La HojadeLiquidacion.TotalAbonar fue calculado.***
- ***Si la liquidación.esConPago se encuentra en false entonces:***

- ***La HojaLiquidación.FechadeBarra fue fijada.***
- ***La HojaLiquidación.Importedebarra fue calculada.***
- ***La HojaLiquidación fue asociada al Generador de Barra (relación formada)***
- ***Las 2 barras fueron creadas***
- ***Generador de Barra fue asociada a la o las barras (relación formada)***

- *La Liquidacion.TotalAbonar fue calculado.*
- ~~*Si Liquidación.esConPago se encontraba en falso, la Liquidación.FechadeBarra fue fijada.*~~
- ~~*Si Liquidación.esConPago se encontraba en falso, la Liquidación.Importedebarra fue calculada.*~~
- ***La HojaLiquidación.RealizoPago es fijada en falso para cada hoja de la liquidación(nuevo atributo).***
- ~~*Si Liquidación.esConPago se encontraba en falso, la Liquidación fue asociada al Generador de Barra.*~~
- ~~*Si Liquidación.esConPago se encontraba en falso, las 2 barras fueron creadas*~~
- ~~*Si Liquidación.esConPago se encontraba en falso, Generador de Barra fue asociada a la o las barras.*~~
- ***Si Liquidación esConPago se encuentra en falso, la Liquidación fue asociada al catálogo de liquidación.***

14.4.3- Revisión el contrato de sistema RealizarPagoOnline.

Nombre:	*RealizarPagos*
Responsabilidades:	*Registrar en el sistema el pago realizado por un contribuyente, realizar el balance e imprimir el recibo.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: : 1.5, 1.6, 1.8* *Casos de Uso: Pagar Impuesto*
Excepciones:	*Si no el importe es menor que el liquidado es un error.*
Pre-condiciones:	*La identificación del dominio es conocido por el sistema.* *El período existe para el vehículo a liquidar.* *Las características del vehículo y los períodos emitidos para ese vehículo son conocidos por el sistema.* *Las liquidaciones realizadas son conocidas por el sistema.*

Post-condiciones:

- ***Si la Respuesta del servicio de Autorización es aprobar el pago:***
  - ***Para Cada período de cada Hoja de Liquidación***
    - *La LíneaPeríodoVehículo.ImportePagado fue modificado (modificación de atributos).*

- *La LíneaPeríodoVehículo.FechaPago fue fijado a la fecha corriente (modificación de atributo).*
- ***Para cada Hoja de Liquidación***
  - *La HojaLiquidación.RealizarPago es fijado en verdadero.*
- ***La liquidación fue asociada al catálogo de liquidación.***

14.4.4- Revisión del Contrato ImputarPagoBatch.

Nombre:	*ImputarPago()*
Responsabilidades:	*Registrar en el sistema el pago realizados por contribuyentes a través del banco.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 1.5, 1.7, 1.8, 2.5, 2.6* *Casos de Uso: CargarPago*
Pre-condiciones:	*El catálogo de Pago ya se encuentra cargado con los datos pendientes de imputación.*

Post-condiciones:

- *Para cada PagoBatch en donde:*

  - *el estado de pago estaba en "I"*
  - *el importecobrado estaba correcto*

  entonces

  - ***Hoja de Liquidación se asoció a PagoBatch.***

  - *el estado de pago estaba en "I"*
  - *el importecobrado estaba correcto*
  - ***se encontró la hoja de liquidación del vehículo con iguales períodos que los reportados en el pagoBatch y que no tenga pago asociado***

  entonces

  - *LíneaPeríodoVehículo.ImportePagado fue modificado para cada período que se pagó.*
  - *LíneaPeríodoVehículo.FechaPago fue modificado para cada período que se pagó.*
  - ***HojaLiquidación.RealizarPago fue fijado en True.***
  - *PagoBatch.estadoDePago fue fijado en "P".*
  - ***La Liquidación se asoció al Catálogo de Liquidación.***

- *Para cada PagoBatch en donde el estado de pago estaba en "I" y donde:*
  - *el importecobrado no estaba correcto*

  o

  - no se encontró hoja de liquidación asociada a Pago

entonces PagoBatch.EstadoPago fue fijado en "E".

14.4.5- Creación del Contrato de HacerPagoTarjetaCrédito.

Nombre:	***HacerPagoTarjetaCrédito*** *(Número Tarjeta: alfa-numérico DNI: alfa-numérico).*
Responsabilidades:	*Crear y requerir autorización de un pago por tarjeta de crédito.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 1.9, 2.2, 2.4. Casos de Uso: Pagar Impuesto*
Output:	*Un requerimiento de pago por tarjeta de crédito fue enviado al servicio de autorización de tarjeta de crédito y se recibe la respuesta del mismo.*
Pre-condiciones:	*El ingreso de periodos a pagar ya ha concluído.*
Post-condiciones:	

- *Un PagoTarjetaCrédito fue creado.*
- *El PagoTarjetaCrédito fue asociado a la Liquidación.*
- *La PagoTarjetaCrédito.ID_Tarjeta fue fijada.*
- *La PagoTarjetaCrédito.ID_DNI fue fijada.*
- *La PagoTarjetaCrédito.ID_Empresa fue fijada.*
- *Un RequerimientoAprobaciónTarjetaCrédito fue creado.*
- *El PagoTarjetaCrédito se asoció al Servicio de Autorización de Tarjeta Crédito con la respuesta del Servicio de Autorización de Tarjeta de Crédito como tipo asociativo.*
- *Si el Servicio de Autorización autoriza el pago:*
  - *Una RespuestaAprobarPagoTarjeta fue creada.*
  - *El PagoTarjetaCrédito fue asociado a Cuenta de Entrantes.*
- *Si el Servicio de Autorización no autoriza el pago:*
  - *Una RespuestaDenegadaPagoTarjeta fue creada.*

14.4.6- Creación del Contrato de HacerPagoTarjetaBanco.

Nombre:	***HacerPagoBanco*** *(Número Tarjeta: alfa-numérico DNI: alfa-numérico).*
Responsabilidades:	*Crear y requerir autorización de un pago por tarjeta de crédito.*
Tipo:	*Sistema*
Referencias Cruzadas:	*Funciones del sistema: 1.9, 2.1, 2.3. Casos de Uso: Pagar Impuesto*
Output:	*Un requerimiento de pago por tarjeta de banco fue enviado al servicio de autorización de tarjeta de banco.*
Pre-condiciones:	*El ingreso de períodos a pagar ya ha concluído.*
Post-condiciones:	

- *Un PagoTarjetaBanco fue creado.*
- *El PagoTarjetaBanco fue asociado a la Liquidación.*
- *La PagoTarjetaBanco.ID_Tarjeta_CBU fue fijada.*
- *La PagoTarjetaBanco.ID_DNI fue fijada.*
- *La PagoTarjetaBanco.ID_Empresa fue fijada.*
- *Un RequerimientoAprobaciónTarjetaBanco fue creado.*
- *El PagoTarjetaBanco se asoció al Servicio de Autorización de Tarjeta Banco con la Respuesta del Servicio de Autorización de Tarjeta de Banco como tipo asociativo.*
- *Si la respuesta fue aprobando el pago por tarjeta:*
  - *Una RespuestaAprobarPagoBanco fue creada.*
  - *El PagoTarjetaBanco fue asociado a Cuenta de Entrantes.*
- *Si el pago por tarjeta fue denegado:*
  - *Una RespuestaDenegadaPagoBanco fue creada.*

Capítulo 15 - Diagramas de colaboración

15.1- Acerca de la organización del capítulo.

En este capítulo, primero revisaremos diagrama de colaboración obtenidos en el primer ciclo de desarrollo para incorporar las nuevas funcionalidades. No se reiterarán las justificaciones de las responsabilidades que se especificaron en el primer ciclo de vida y que no han variado, solo estudiaremos los cambios en el diagrama.

Luego examinaremos nuevas operaciones de sistema que han aparecido en este ciclo.

En ambos casos, utilizaremos los patrones GRASP[Larman97] (ver Anexo "Patrones Grasp") para justificar las decisiones tomadas en cada diagrama.

Dividiremos los diagramas de colaboración cuando consideremos que se tornan complejos.

Nota: No hemos mencionado el diagrama de colaboración de CarpaPagoBatch e Inicializar porque no sufrieron modificaciones respecto al primer ciclo

15.2- Diagrama de colaboración de la operación de sistema Entrar Período.

En este diagrama de colaboración debemos incorporar la restricción de que no puede haber más de 4 períodos por hoja de liquidación. Mirando el modelo conceptual del capítulo 14, podemos deducir que un Catálogo de Liquidación contiene Liquidaciones, las que a su vez contiene Hojas de Liquidación.

Como ya justificamos anteriormente por patrón experto, Terminal tendrá la responsabilidad de crear una nueva liquidación (es en ella donde se registran). Sin embargo, a diferencia con el primer ciclo, Liquidación no creará la colección de LíneaPeríodoLiquidado, dado que sus instancias ya no mantienen dicha colección. Lo que ahora mantienen son las HojasLiquidación, y esta es la colección que generará.

Por otro lado, consideremos las siguientes post-condiciones del contrato EntrarPeríodo:

- *Una HojaLiquidación fue creada*
- *La HojaLiquidación fue asociada a la Liquidación*

La primera de estas post-condiciones implica una responsabilidad de creación y dado que una Liquidación contiene 1 o varias HojasLiquidación es razonable pensar que es la clase candidata apropiada para crear una nueva hoja (usamos patón creador).

En este momento, también deberá crearse una colección vacía para registrar los períodos que serán ingresados. Dado que dicha colección será mantenida por una instancia de HojaLiquidación, ella es la candidata para tener la responsabilidad de la creación (por patrón creador).

Luego de la generación de una Liquidación y de una Hoja, se deberá proceder al ingreso de un nuevo PeríodoLineaLiquidado. Terminal tendrá la responsabilidad de solicitárselo a Liquidación (dado que posee visibilidad sobre ella) enviándole el mensaje HacerLíneaPeríodoLiquidado. Este último mensaje lo estudiaremos en la próxima sección.

El diagrama de colaboración principal queda así de la siguiente manera:

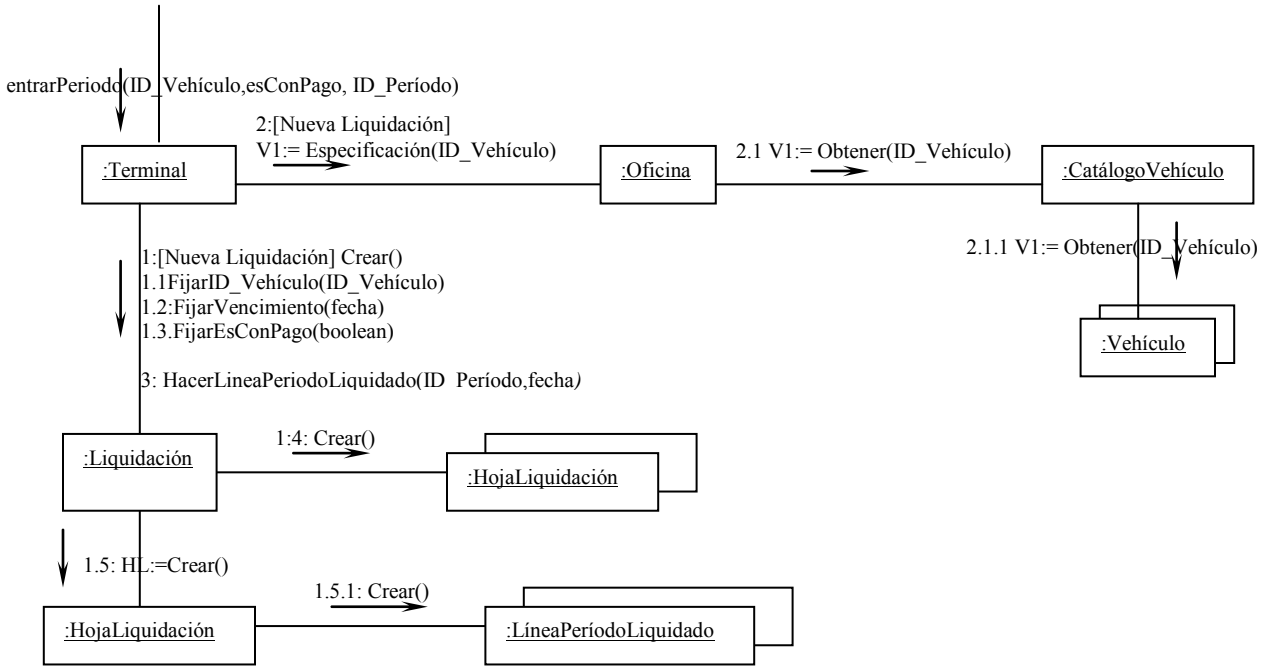


Figura 15.1
Diagrama de colaboración de entrarPeríodo

Aclaración: Hemos decidido que Terminal le solicite los datos del Vehículo a Oficina y no a Catálogo, porque creemos que ya tiene suficientes responsabilidades como para agregarle la que implica la visibilidad sobre el Catálogo de Vehículo.

15.2.1- Operación HacerLíneaPeríodoLiquidado.

Mirando el contrato de sistema podemos denotar que cuando una hoja de liquidación posee 4 períodos en su colección, entonces ocurren las siguientes post-condiciones:

1- *La HojadeLiquidacion.TotalAbonar fue calculado.*
2- *La HojaLiquidación.FechadeBarra fue fijada.*
3- *La HojaLiquidación.Importedebarra fue calculada.*
4- *La HojaLiquidación fue asociada al Generador de Barra (relación formada).*
5- *Las 2 barras fueron creadas.*
6- *Una hoja de Liquidación fue creada.*
7- *La Hoja de liquidación fue asociada a la liquidación*

Las post-condiciones de la 1 a la 5 realizan una serie de tareas relacionadas a la completación de la HojadeLiquidación con la que veníamos trabajando. Estas tareas fueron separadas en otro diagrama de colaboración inicializado con el mensaje TerminarHojaLiquidación. Lo que aquí nos ponemos a pensar, es quien tendrá la responsabilidad de realizar la tarea de completación de una Hoja de Liquidación. Dado que en este proceso calcularemos atributos de la Hoja de Liquidación, y que es la clase la

responsable del mantenimiento de sus atributos; por experto concluimos que Liquidación le enviará un mensaje a HojaLiquidación, quien tendrá la responsabilidad de realizar la tarea.

La post-condición del punto 6 constituye una responsabilidad de creación. Puesto una Liquidación posee 1 o varias HojaLiquidación, luego Liquidación es (por patrón creador) la clase indicada para crear una nueva HojaLiquidación. En el momento de la creación, se generará una colección de LíneaPeríodoLiquidado, responsabilidad que daremos a HojaLiquidación dado que la mantiene.

Estudiemos ahora este otro conjunto de post-condiciones del diagrama de colaboración

1. *Una LíneaPeríodoLiquidado fue creada.*
2. *La Hoja de Liquidación fue asociada a la nueva LíneaPeríodoLiquidado.*
3. *La LíneaPeríodoLiquidado.PeríodoLiquidado fue fijado.*
4. *La LíneaPeríodoLiquidado.VencimientoOriginal fue fijado.*
5. *La LíneaPeríodoLiquidado.DeudaOriginal fue fijado.*

La primera de todas estas post-condiciones denota una responsabilidad de creación.

La pregunta que surge es ¿Quién tendrá la responsabilidad de crear una nueva instancia de la LíneaPeríodoLiquidado?

Puesto que una HojaLiquidación contiene 1 a 4 LíneaPeríodoLiquidado, constituye por patrón creador un candidato natural para tener tal responsabilidad y durante la creación le mandará los datos de inicialización de sus atributos.

Las post-condiciones que se encuentran entre los puntos 3 a 5 constituye la fijación de atributos de la clase LíneaPeríodoLiquidado. Luego, por experto (dado que una clase mantiene sus atributos), podemos deducir que LíneaPeríodoLiquidado es la clase indicada para tener tal responsabilidad.

Para realizar la tarea de inicialización de atributos, LíneaPeríodoLiquidado requiere de los valores de inicialización. Dichos valores se encuentran en la Línea períodoVehículo. Existen dos alternativa a seguir:

- *La líneaPeríodoLiquidado puede solicitarle a la LíneaPeríodoVehículo, los datos que necesita. Al hacer esto estaríamos violando el patrón Grasp de "No hablar con extraños", ya que estas dos clases no tienen visibilidad entre sí (esto lo logramos deducir mirando el modelo conceptual). Esta solución provocaríamos que LíneaPeríodoLiquidado tenga conocimiento de LíneaPeríodoVehículo, lo cual hace a la dependencia de la clase y por lo tanto genera un mayor acoplamiento entre clases. Por lo expuesto, no creemos que esta sea una solución viable.*
- *Segunda alternativa:*
  - *Liquidación ya tiene visibilidad sobre Vehículo, luego Liquidación le solicitará a Vehículo los datos necesarios para la creación de la nueva LíneaPeríodoLiquidado.*
  - *Vehículo obtendrá el período que se necesita de la colección LíneaPeríodoVehículo(que él mantiene y por ende tiene visibilidad)*

en base la identificación de período y tendrá la responsabilidad de solicitarle a tal instancia los datos deseados.

- *La instancia de la LíneaPeríodoVehículo será la responsable de devolver los datos solicitados por Vehículo(por experto).*
- *La liquidación, por otra parte, tiene visibilidad sobre la HojaLiquidación (ya que ella las contiene), luego le enviará los datos de inicialización a la misma.*
- *La HojaLiquidación contiene las LíneaPeríodoLiquidado, entonces ella le enviará los datos a la misma (por experto).*

Esta última alternativa nos parece más interesante, ya que no provocamos dependencias innecesarias, haciendo que clases que hasta ahora no tenían conocimiento una de otra, ahora lo tengan. Además respetamos el principio de encapsulamiento de las clases, ya que, son las clases contenedoras de otras las que solicitan datos. Lo que queremos decir es:

En el ejemplo decidimos que Vehículo le solicite los datos a PeríodoLíneaVehículo y no LíneaPeríodoLiquidado. Vehículo es la clase que contiene todas las LíneasPeríodoVehículo. Luego, si es ella y no un extraño (LíneaPeríodoLiquidado) quien solicita la información, lograremos respetar el principio de encapsulamiento de datos.

El diagrama de colaboración quedará de la siguiente manera:

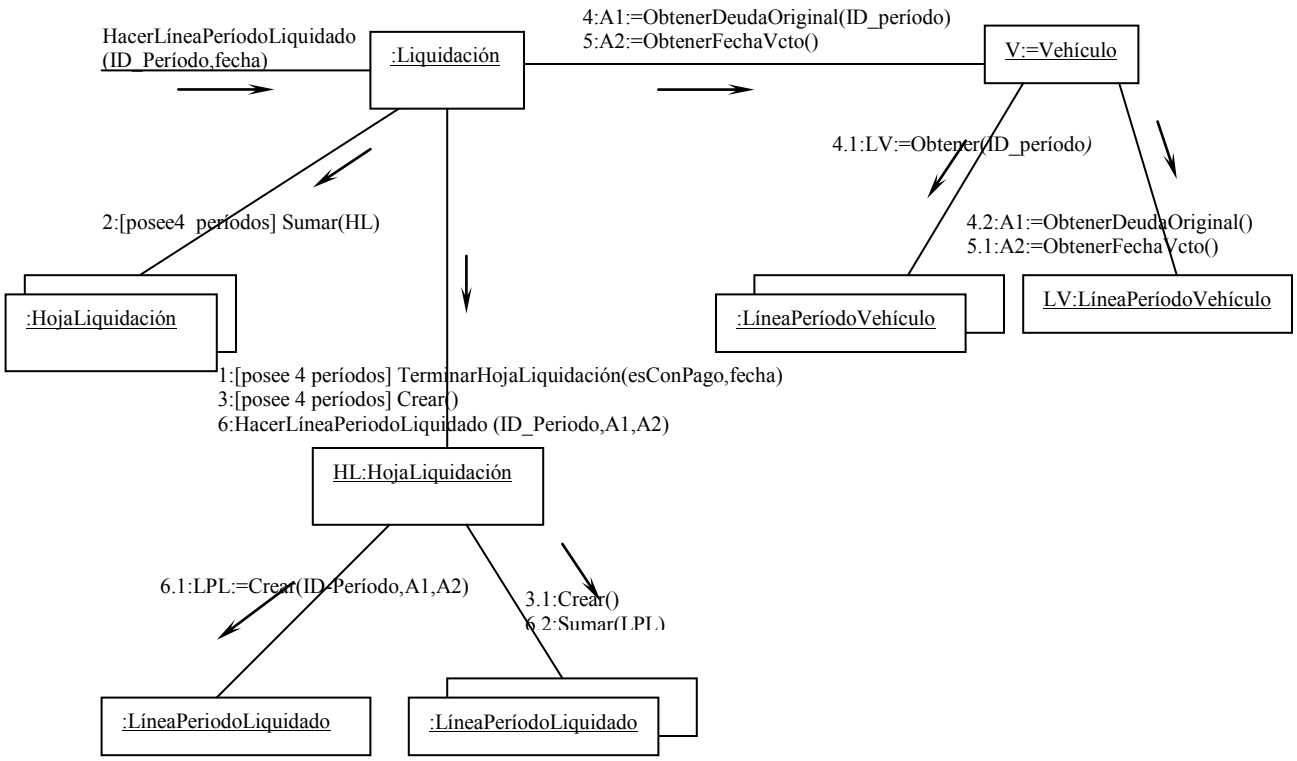


Figura 15.2
Diagrama de colaboración de hacerLíneaPeríodoLiquidado.
Llamada en la operación entrarPeríodo.

15.2.2- Operación TerminarHojaLiquidación.

En el contrato de la operación de sistema EntrarPeríodoLiquidar aparecen las siguientes post-condiciones:

- *Si la cantidad de períodos en la hoja es de 4*
  - *La HojadeLiquidacion.TotalAbonar fue calculado*
  - *La HojadeLiquidación.FijarPago fue fijada*
- *Si la cantidad de períodos en la hoja es de 4 y la liquidación.esConPago se encuentra en false entonces:*
  - *La HojaLiquidación.FechadeBarra fue fijada.*
  - *La HojaLiquidación.Importedebarra fue calculada.*
  - *La HojaLiquidación fue asociada al Generador de Barra (relación formada)*
  - *Las 2 barras fueron creadas*

Mirando estas post-condiciones podemos concluir que, dado que HojaLiquidación posee la información necesaria para cumplir las tareas, por patrón experto (dado que son atributos propios que ella debe mantener), tendrá la responsabilidad de:

- *Fijar el atributo RealizarPago*
- *Calcular y fijar el atributo de TotalAbonar*
- *Fijar el atributo fechadeBarra*
- *Calcular y fijar el atributo de ImportedeBarra*

Nota: Estas 2 últimas tareas se realizarán en caso de cumplirse las condiciones establecidas.

Por otro lado, la última post-condición expuesta, denota una responsabilidad de creación, que será por patrón creador, realizada por el Generador de Barras. Para dicha creación, se deberá enviar la decodificación de la barra. La HojadeLiquidación es responsable, por experto, de su generación dado que posee todos los datos necesarios para su cálculo. Una vez calculada dicha decodificación es enviada, vía parámetro al generador de barras.

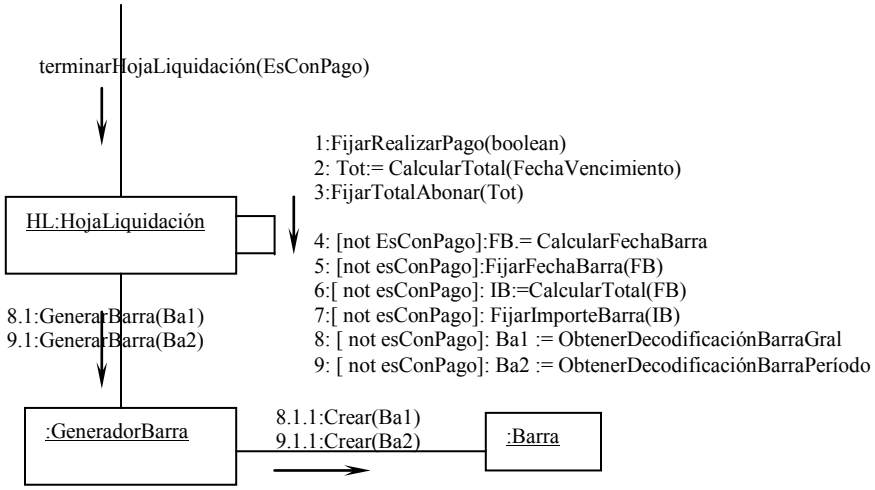


Figura 15.3
Diagrama de colaboración de terminarHojaLiquidación.
Llamada en las operaciones entrarPeríodo y terminarLiquidación

15.2.3 - Operación CalcularTotal de la Hoja de Liquidación.

En el primer ciclo de desarrollo, para realizar esta operación le solicitabamos a la Liquidación que devolviera todos las instancias de LíneasPeríodoLiquidado, para que dichas instancias, a su vez, devolvieran el importe actualizado de los períodos liquidado.

Ahora es la HojaLiquidación la que posee las LíneasPeríodoLiquidado y no la Liquidación, luego será ella quien obtendrá todas las instancias de su colección y le solicitará los datos necesarios. Las LíneaPeríodoLiquidado tendrá, por experto, la responsabilidad de devolver el importe actualizado, para que HojaLiquidación lo sume.

Para mayor explicación sobre la forma en que surge esta operación común a total abonar y al importe de la barra remitirse a la explicación del diagrama de colaboración del primer ciclo de desarrollo de esta tesis (capítulo 7, sección 7.10.1).

El diagrama de colaboración quedará de la siguiente manera:

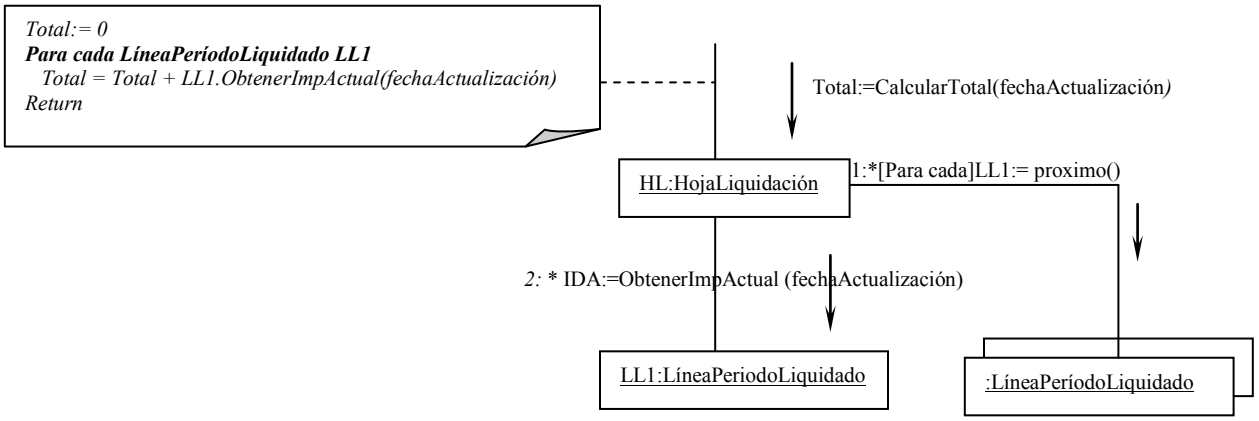


Figura 15.4
Diagrama de colaboración de calcularTotal.
Llamada en la operación terminarHojaLiquidación.

15.3- Diagrama de colaboración de la operación de sistema TerminarLiquidación.

Cuando el operador informa al sistema que no se ingresarán más períodos es necesario que se complete la última HojaLiquidación con la que estabamos trabajando. Por experto, Hoja de Liquidación tiene tal responsabilidad y lo hace cuando recibe el mensaje TerminarHojaLiquidación (explicado en la sección 15.2.2 de este capítulo). Nuevamente, por experto, Liquidación tendrá la responsabilidad de envío del mensaje.

Además, será necesario que se guarde la última hojaLiquidación en la colección de hojaLiquidación. Dado que Liquidación mantiene dicha colección será ella la que tendrá la responsabilidad de realizar tal tarea.

Mirando el contrato de la operación observamos la siguiente post-condición:

- *La Liquidacion.TotalAbonar fue calculado*

Esta post-condición indica la responsabilidad de calcular el TotalAbonar de toda la liquidación. Para ello Liquidación obtendrá cada una de las instancias de la colección HojaLiquidación. HojaLiquidación, por experto, tendrá la responsabilidad de devolverle el valor de su atributo TotalAbonar, para que Liquidación lo sume y así obtenga el Total a abonar de toda la liquidación.

El diagrama de colaboración tendrá la siguiente forma:

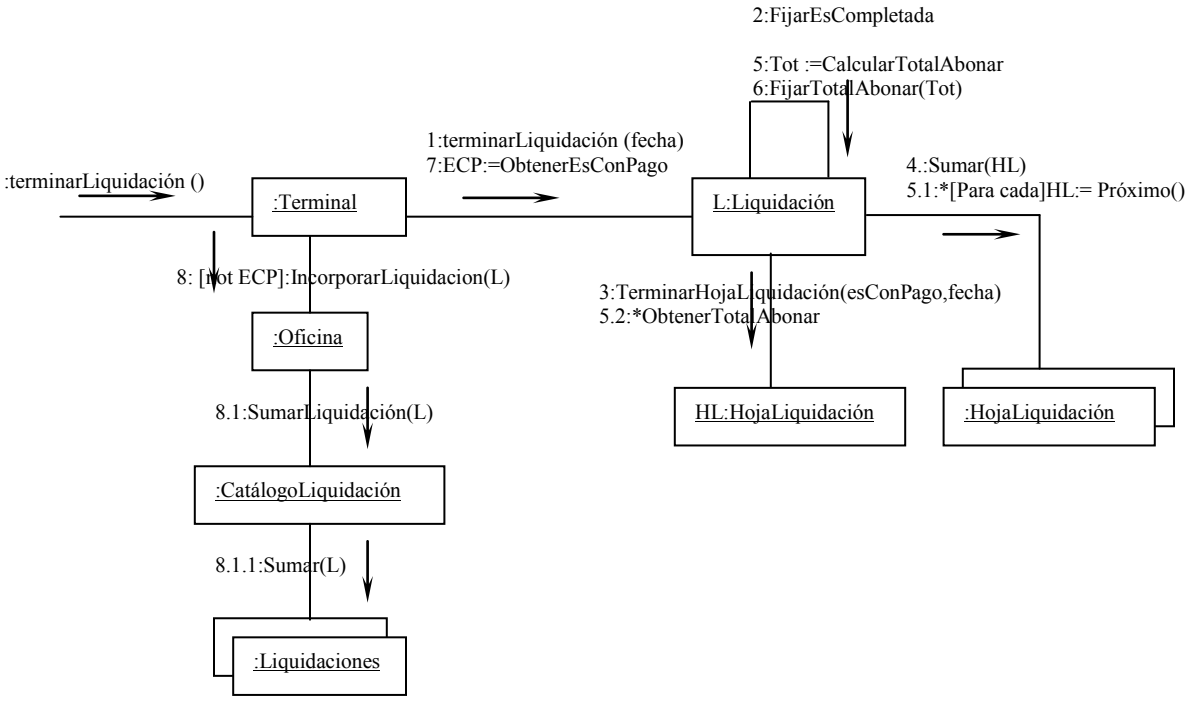


Figura 15.5
Diagrama de colaboración de la operación de sistema terminarLiquidación.

Nota: *El almacenaje de la Liquidación en el Catálogo de Liquidación se realizará solo si no se realiza el pago en el momento. Esto se debe a que en caso de realizarse el pago mediante tarjeta, deberá solicitarse la autorización de la misma (pudiendo ser denegada en cuyo caso la liquidación será cancelada y no deberá ser guardada en ningún lugar).*

15.4 - Diagrama de colaboración de RealizarPagoOnline.

Mirando el contrato de esta operación vemos que, cuando se realiza el pago online ya sea con tarjeta de crédito o tarjeta bancaria, el pago corresponde a toda la liquidación. En cambio, los pagos que son ingresados por banco (en forma Batch) corresponden a una única hoja de liquidación. Luego a diferencia de ImputarPagoBatch en donde se imputa una HojaLiquidación, se deberá imputar todas las HojasLiquidación que se encuentren en la colección contenida en Liquidación.

Así el diagrama de colaboración quedará de la siguiente manera:

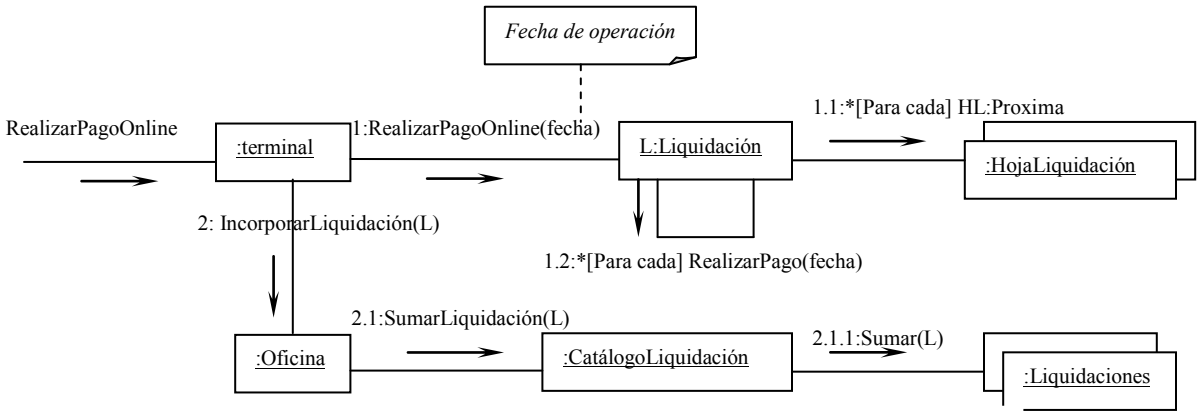


Figura 15.6
Diagrama de colaboración de la operación de sistema realizarPagoOnline.

En este diagrama vemos que Terminal tendrá la responsabilidad de solicitarle a Liquidación de que realice el pago online.

Dado que Liquidación contiene cada una de las HojaLiquidación en la colección HojaLiquidación, por experto, será responsable de obtener cada una de dichas Hojas.

La pregunta que nos surge es ¿quién es el responsable de realizar el proceso de imputación propiamente dicho de todos los períodos de una HojaLiquidación? (es decir de sumar el importe liquidado en cada período al importe pagado del vehículo):

- *Una alternativa sería que HojaLiquidación tuviera la responsabilidad de realizar tal tarea ya que ella contiene las líneas de períodos que se desean imputar. Pero si esta clase tiene tal responsabilidad debería tener visibilidad sobre la clase Vehículo ya que ella posee la colección de períodos del vehículo. Esto aumentaría el acoplamiento y la dependencia entre ambas clase y no la consideramos, por ende una alternativa fiable.*
- *Por otro lado, observando el modelo conceptual vemos que Liquidación ya tiene visibilidad sobre el Vehículo y sobre las HojasLiquidación. Dado que la elección de esta clase para la realización de tal tarea no aumentará el acoplamiento, ni disminuiría la cohesión, la consideramos una buena alternativa.*

En la sección subsiguiente mostraremos el diagrama de colaboración de RealizarPago. Dicha operación no solo será usada aquí sino en el proceso de imputación de los pagos provenientes del banco.

Por último, se realizará la tarea de almacenaje de la Liquidación en el Catálogo de Liquidación.

***Nota:** Si llegamos a realizar esta operación de sistema es porque estamos seguros que el pago está autorizado, luego la Liquidación siempre debe ser guardada.*

15.4.1- Diagrama de colaboración RealizarPago.

Cuando Liquidación recibe el mensaje de Realizar el pago de la HojaLiquidación seleccionada, tendrá la responsabilidad por experto de:

- *Solicitarle a la HojaLiquidación la identificación del período y la deuda original de cada uno de sus objetos contenidos en la colección LíneaPeríodoLiquidado.*
- *Solicitarle a Vehículo que modifique los datos de pago de cada período con la identificación de la parámetro de entrada.*
- *Solicitarle a HojaLiquidación que fije su atributo RealizarPago en true.*

HojaLiquidación será, por su parte, responsable de (por expertos y dado que ella contiene las LíneasPeríodoLiquidado):

- *Obtener la próxima LíneaPeríodoLiquidado de su colección.*
- *Solicitarle a la LíneaPeríodoLiquidado que devuelva el valor de su identificación de período y la deuda original.*
- *Modificar su atributo RealizarPago.*

LíneaPeríodoLiquidado tendrá la responsabilidad por experto (es responsable de mantener sus atributos) de:

- *Devolver el valor de la identificación de período y la deuda original*

Vehículo, tendrá la responsabilidad de (por expertos y dado que contiene las LíneasPeríodoVehículo):

- *Obtener de su colección la LíneaPeríodoVehículo cuya identificación de vehículo es igual a la de parámetro*
- *Solicitarle a la instancia de LíneaPeríodoVehículo que modifique sus atributos de acuerdo a los demás datos de parámetro.*

Por último, LíneaPeríodoVehículo, tendrá la responsabilidad de:

- *Modificar sus atributos fecha de pago e importe pagado de acuerdo a los datos de entrada*

El diagrama de colaboración quedará de la siguiente forma:

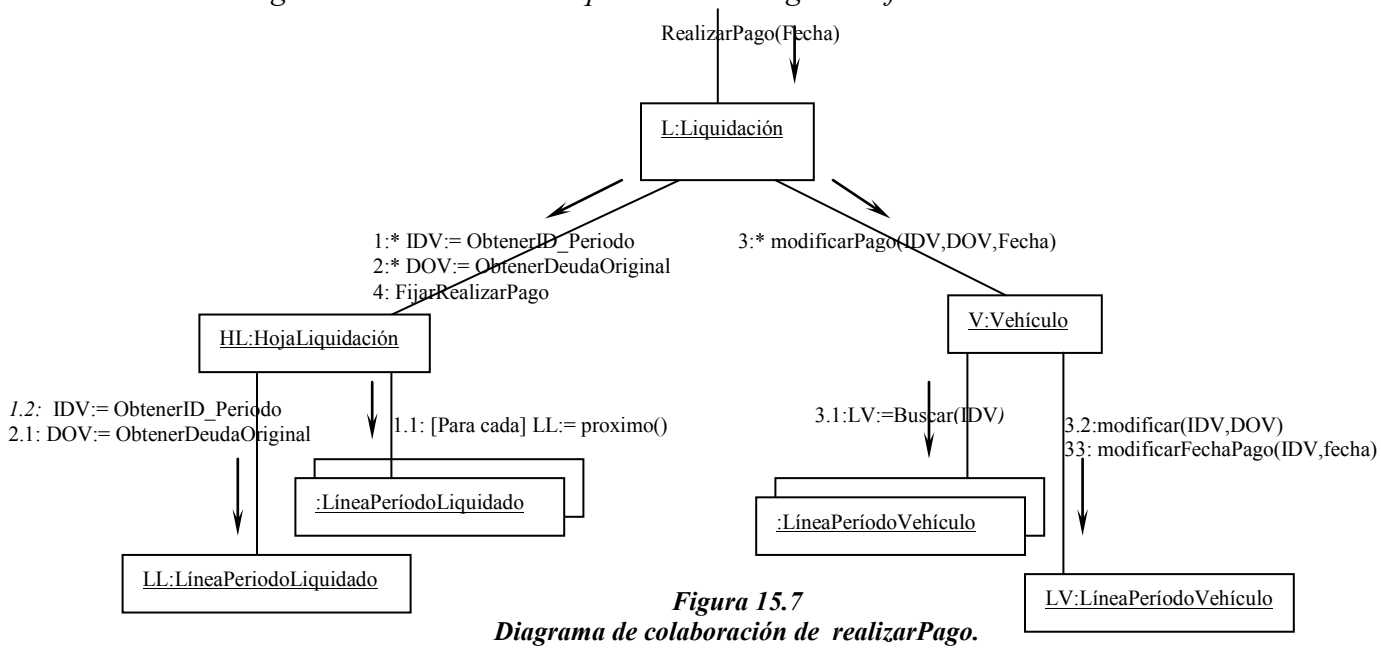


Figura 15.7
Diagrama de colaboración de realizarPago.
Llamada en realizarPagoOnline y en imputarSiguiente

15.5- Diagrama de colaboración de la operación de sistema ImputarPagoBatch.

El diagrama de colaboración principal de esta operación no cambia, pero sí la operación ImputarSiguiente.

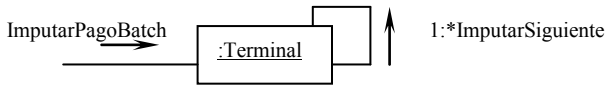


Figura 15.8
Diagrama de colaboración de operación de sistema imputarPagoBatch

15.5.1- Diagrama de colaboración de la operación ImputarSiguiente.

Según lo desarrollado en el primer ciclo, en la sección de Imputación de pago del capítulo8, lo primero que realizaremos es obtener el próximo PagoBatch pendiente de imputación y validarlo.

Luego se deberá terminar el proceso de validación del PagoBatch, teniendo en cuenta que se deberá comparar con una HojaLiquidación y no con una Liquidación como en el primer ciclo de vida:

1- *Buscar la próxima HojaLiquidación para el dominio al que se le haya realizado el pago: Como se ve en el modelo conceptual, el Catálogo de Liquidación contiene Liquidaciones y a su vez cada Liquidación contiene HojasLiquidación. Luego, dado que la oficina tiene visibilidad sobre el Catálogo de Liquidación, ella le enviará un mensaje para solicitarle al mismo que devuelva la próxima Hoja de Liquidación del vehículo especificado.*
2- *Comparar el PagoBatch obtenido con una HojaLiquidación obtenida en el punto 1.*

En la sección subsiguiente estudiaremos más profundamente como se realiza las dos tareas especificadas en los puntos de arriba.

Por último se procederá al proceso de imputación o de marcado de error de acuerdo al resultado de la validación.

Así el diagrama de colaboración queda con el siguiente formato:

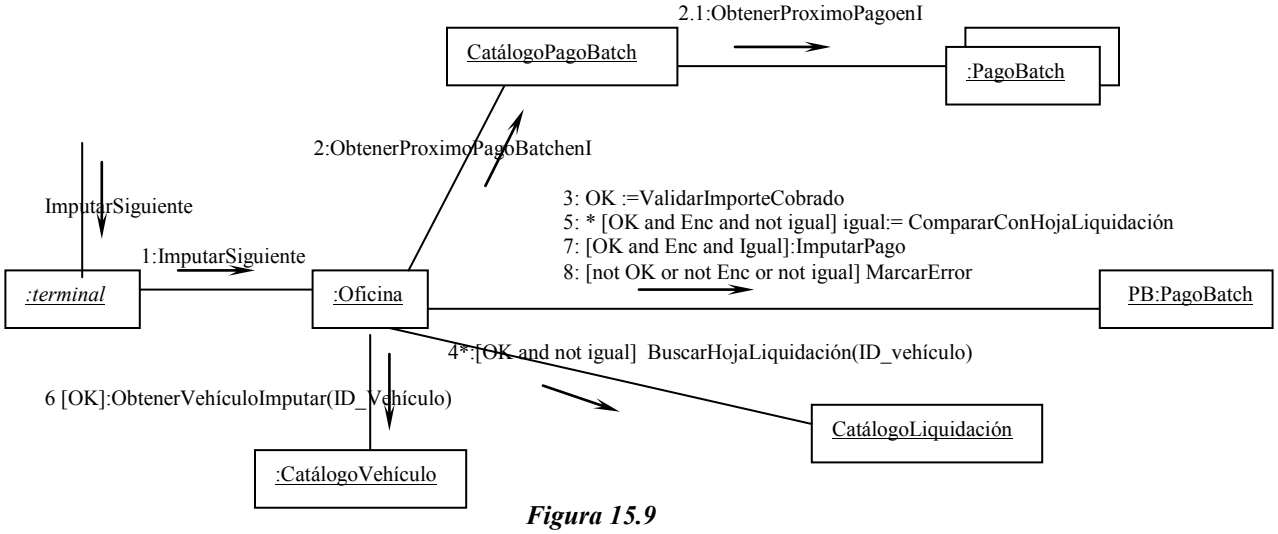


Figura 15.9
Diagrama de colaboración de operación de sistema imputarSiguiente – Versión 1

Este diagrama de colaboración no nos satisface demasiado ya que Oficina tiene visibilidad sobre PagoBatch. Esta visibilidad hace al aumento del acoplamiento y a la disminución de la cohesión de la clase(en Oficina).

Luego hemos pensado un diagrama alternativo

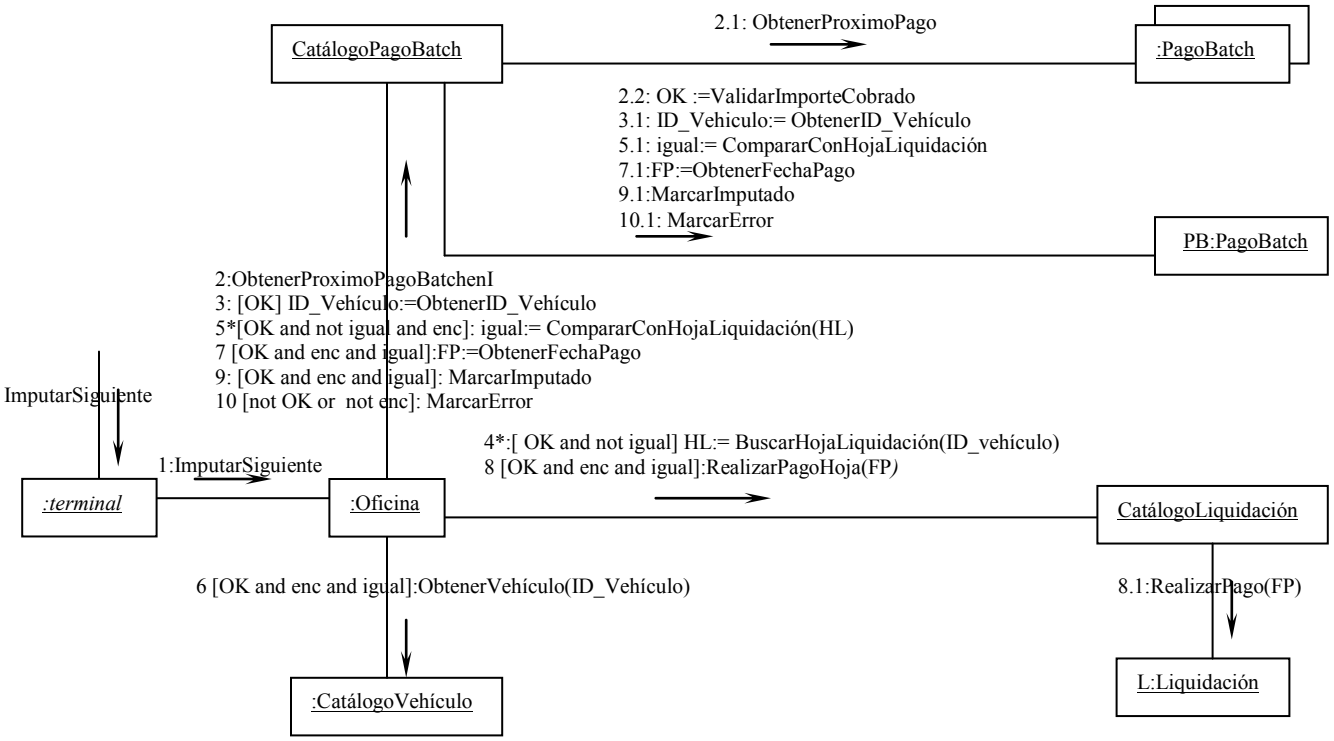


Figura 15.10
Diagrama de colaboración de operación de sistema imputarSiguiente – Versión 2

Este diagrama de colaboración realiza las mismas tareas que el anterior, pero nos parece que posee una asignación de responsabilidades más correcta ya que provoca menos acoplamiento entre las clases y mayor cohesión respecto al anterior.

La secuencia de acciones a seguir será la siguiente:

- *Terminal le solicitará a Oficina que impute el siguiente pago. Oficina es, por experto, la clase adecuada para realizar tal tarea ya que posee toda la información necesaria.*
- *Para realizar solicitado por Terminal, Oficina deberá:*
  1. *Solicitarle a Catálogo de PagoBatch el próximo pago Batch pendiente de imputación.*
  2. *Solicitarle a Catálogo de Liquidación tantas HojasLiquidación necesarias hasta encontrar la que se corresponda con el PagoBatch encontrado en el punto 1.*
  3. *Solicitarle a Catálogo de PagoBatch que compare la HojaLiquidación encontrada en el punto 2 y el PagoBatch encontrado en punto 1 (para ver si efectivamente se corresponden).*

4. *En caso de que la Validación resultase correcta:*
   4.1. Solicitarle a Catálogo de Vehículo, que obtenga el Vehículo al que se le efectuará la imputación de períodos.
   4.2. Solicitarle a Catálogo de Pago Batch, que obtenga la Fecha de Pago
   4.3. Solicitarle a Catálogo de Liquidación que realice el pago de la HojaLiquidación obtenida en el punto 2. Notemos que aquí Catálogo de Liquidación enviará el mensaje realizarPago a Liquidación (mensaje ya explicado en la sección 15.4.1 de este capítulo).
   4.4. Solicitarle a Catálogo de Pago Batch que marque al PagoBatch como imputado.
5. *En caso de que la validación resultase incorrecta, solicitarle al Catálogo de PagoBatch que marque el pago como erróneo.*

Nota: *Como vemos, Oficina ya no necesita tener visibilidad sobre PagoBatch, ella solo envía mensajes a objetos que contiene. De esta manera disminuimos el acoplamiento de la clase. Además aumentamos su cohesión, ya que posee responsabilidades más relacionadas entre sí, dado que la tarea de mantenimiento del PagoBatch no tenía mucho que ver con las otras funcionalidades de la clase.*

- *Como vemos en el modelo conceptual CatálogoPagoBatch contiene PagosBatch, luego será responsable, por patrón experto de:*
  1. *Buscar en su colección el próximo PagoBatch pendiente de imputación.*
  2. *Solicitarle a la instancia de PagoBatch que realice las tareas que requiera Oficina (según las condiciones que se establezcan).*

- *PagoBatch será responsable, por patrón experto de:*
  1. *Validar el atributo supuesto cobrado que ella posee.*
  2. *Devolver el valor de su atributo de ID_Vehículo, necesario para la obtención de la Hoja de Liquidación con que se comparará.*
  3. *Compararse con la HojaLiquidación (mediante la operación CompararconHojaLiquidación explicada en el punto 15.5.3), devolviendo un valor indicativo del resultado de la operación (true o false).*
  4. *Devolver el valor de su atributo de fecha de pago, necesario para completar el atributo fecha de pago en LíneaPeríodoVehículo.*
  5. *Dependiendo del resultado de la validación, se marca como imputado o erróneo, completando el atributo estado pago en "P" o "E" según corresponda.*

- *CatálogoLiquidación será responsable de:*
  1. *Buscar la próxima Hoja de Liquidación a ser comparada con el PagoBatch, según la operación BuscarHojaLiquidación estudiada en la sección 15.5.2 de este capítulo).*
  2. *Realizar la imputación de la HojaLiquidación corriente. Para ello enviará el mensaje RealizarPago ya utilizado en el momento de imputación de Pago Online (ver sección 15.4.1).*

15.5.2- Diagrama de colaboración de BuscarHojaLiquidación.

Cuando la oficina de liquidación, le envíe le mensaje a CatálogoLiquidación, él tendrá la responsabilidad (por experto) de:

- *Obtener de su colección de Liquidaciones la próxima Liquidación con identificación de vehículo igual que la ingresada por parametro (en caso de que la liquidación en proceso no posea más hojas).*
- *Solicitarle a la Liquidación que devuelva la próxima hoja de su colección.*

Liquidación por su parte, dado que ella mantiene la colección de HojasLiquidación tendrá la responsabilidad de devolver la próxima hoja de su colección que no tiene pago asociado..

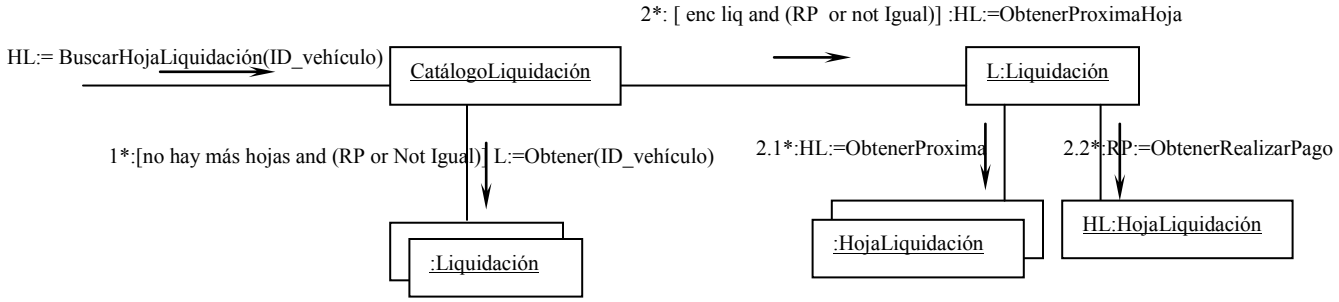


Figura 15.11
Diagrama de colaboración de buscarHojaLiquidación
Llamada por imputarSiguiente

15.5.3- Diagrama de CompararconHojaLiquidación.

La única diferencia que tiene esta operación con respecto a la del primer ciclo, es que la comparación del PagoBatch no se realizará contra una instancia de la Liquidación, sino contra una instancia de la HojaLiquidación. Esto se debe a que esta operación involucra comparaciones de atributos que ahora se encuentran en la HojaLiquidación y no en la Liquidación, tales como el importe de la barra, fecha de la barra, las líneasPeríodoLiquidado, etc.

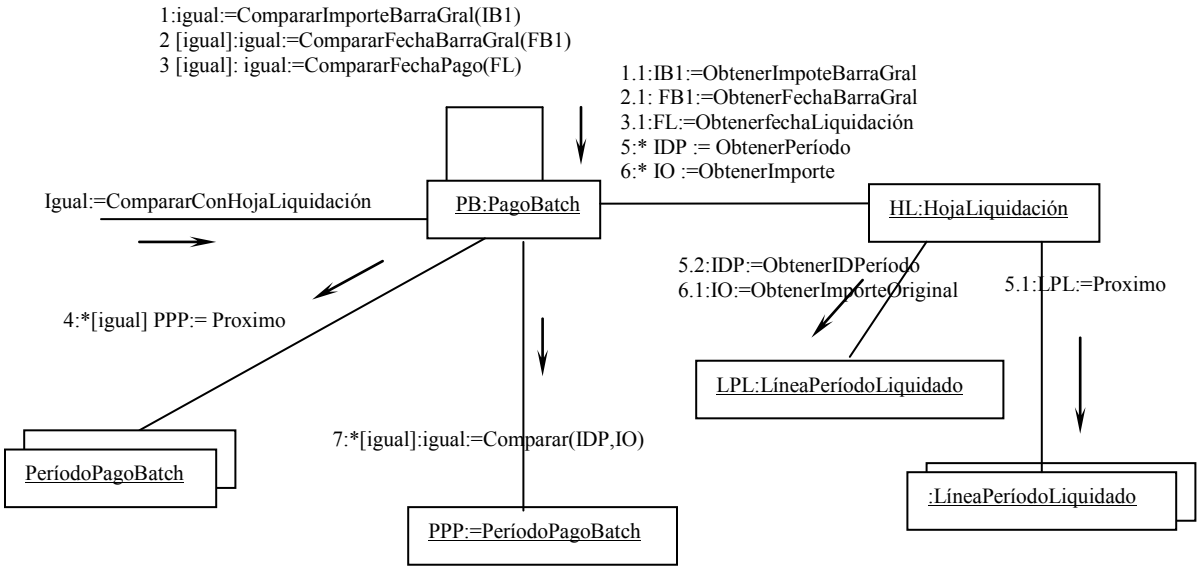


Figura 15.12
Diagrama de colaboración de compararConHojaLiquidación
Llamada por imputarSiguiente

15.6- Polimorfismo en las operaciones de sistema HacerPagoTarjetaCrédito y HacerPagoTarjetaBanco.

Mirando el diagrama conceptual notamos que existen alternativas basadas en tipos (clases). Es decir, dependiendo del tipo de pago que se realizará (pago por tarjeta de crédito o tarjeta de banco) es el comportamiento que tendrá nuestro sistema.

El patrón Grasp de polimorfismo [Larman97], sugiere que cuando ocurren estas situaciones, se debe asignar responsabilidades por comportamiento (utilizando operaciones polimórficas) al tipo para el cual varía el comportamiento. En nuestro caso, dado que el comportamiento de la autorización varia de acuerdo a la clase de pago, luego por polimorfismo deberíamos asignar la responsabilidad de autorización al tipo de pago, implementado con una ***operación autorizar polimórfica.*** *El esquema se vería de la siguiente manera:*

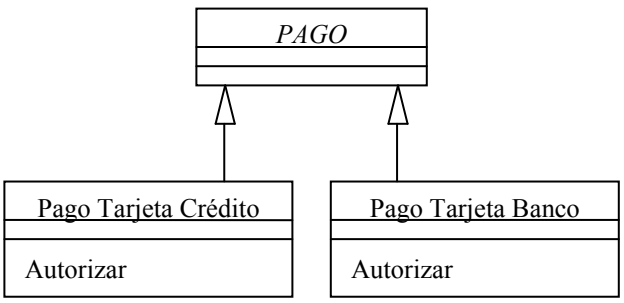


Figura 15.13

15.7- Creación del diagrama de colaboración HacerPagoTarjetaCrédito.

Observemos las siguientes post-condiciones del contrato de la operación de sistema:

- *Un PagoTarjetaCrédito fue creado.*
- *El PagoTarjetaCrédito fue asociado a la Liquidación.*
- *La PagoTarjetaCrédito.ID_Tarjeta fue fijada.*
- *La PagoTarjetaCrédito.ID_DNI fue fijada.*
- *La PagoTarjetaCrédito.ID_Empresa fue fijada.*

La primera post-condición indica una responsabilidad de creación. Nuestra pregunta es ¿quién tendrá la responsabilidad de realizar la creación de una nueva instancia de PagoTarjetaCrédito?.

- *Terminal es una clase candidata dado que en ella se registran los pagos por tarjeta de crédito. Sin embargo, si le asignamos a esta clase la responsabilidad de creación, Terminal deberá tener conocimiento de PagoTarjetaCrédito (lo cual hasta ahora no tenía).*
- *Por otro lado otra clase candidata para tener la responsabilidad de crear el PagoTarjetaCrédito y asociarlo a la Liquidación sería Liquidación. Esta alternativa provoca menos acoplamiento entre las clases dado que Liquidación debe tener conocimiento de PagoTarjetaCrédito ya sea la cree o no. Luego consideramos esta opción más apropiada.*

En el momento de creación, Liquidación mandará los parámetros de inicialización de atributos a PagoTarjetaCrédito.

Luego de la creación del PagoTarjetaCrédito, Liquidación tendrá enviarle un mensaje para que autorice el pago. Como ya explicamos en la sección anterior este mensaje será polimórfico (existirá un mensaje con igual nombre en PagoTarjetaBanco y dependiendo del tipo de entrada será el mensaje que se ejecute).

Observando la siguiente post-condición del contrato y el diagrama conceptual:

- *El PagoTarjetaCrédito se asoció al Servicio de Autorización de Tarjeta Crédito con la respuesta del Servicio de Autorización como tipo asociativo.*

Podemos deducir que, se requerirá el acceso a una instancia del Servicio de Autorización de Tarjeta de Crédito, luego el sistema debe comunicarse con un servicio externo. Cuando nos encontramos en esta situación, el patrón GoF de Proxy Remoto sugiere realizar una clase de software local que representa al servicio externo y darle la responsabilidad de conexión con el servicio externo real. Esta clase evita el acoplamiento en nuestro sistema, constituyendo una indirección, según patrón Grasp.

En nuestro caso, deberemos encontrar el servicio de autorización al que deseamos solicitar la autorización (basados en el nombre de la empresa) y luego utilizando el Proxy Remoto deberemos comunicarnos con el servicio de autorización real.

La clase Oficina tiene conocimiento sobre los servicios de autorización, luego por experto, tendrá la responsabilidad de devolver el Proxy de autorización de servicio de tarjeta, cuando la clase PagoTarjetaCrédito se lo solicite.

Una vez que se encuentra el servicio de autorización correcto, él tendrá la responsabilidad de completar la autorización, para lo cual necesita enviar el mensaje de requerimiento de aprobación fuera del sistema y recibir la respuesta del servicio externo.

El diagrama se verá de la siguiente manera:

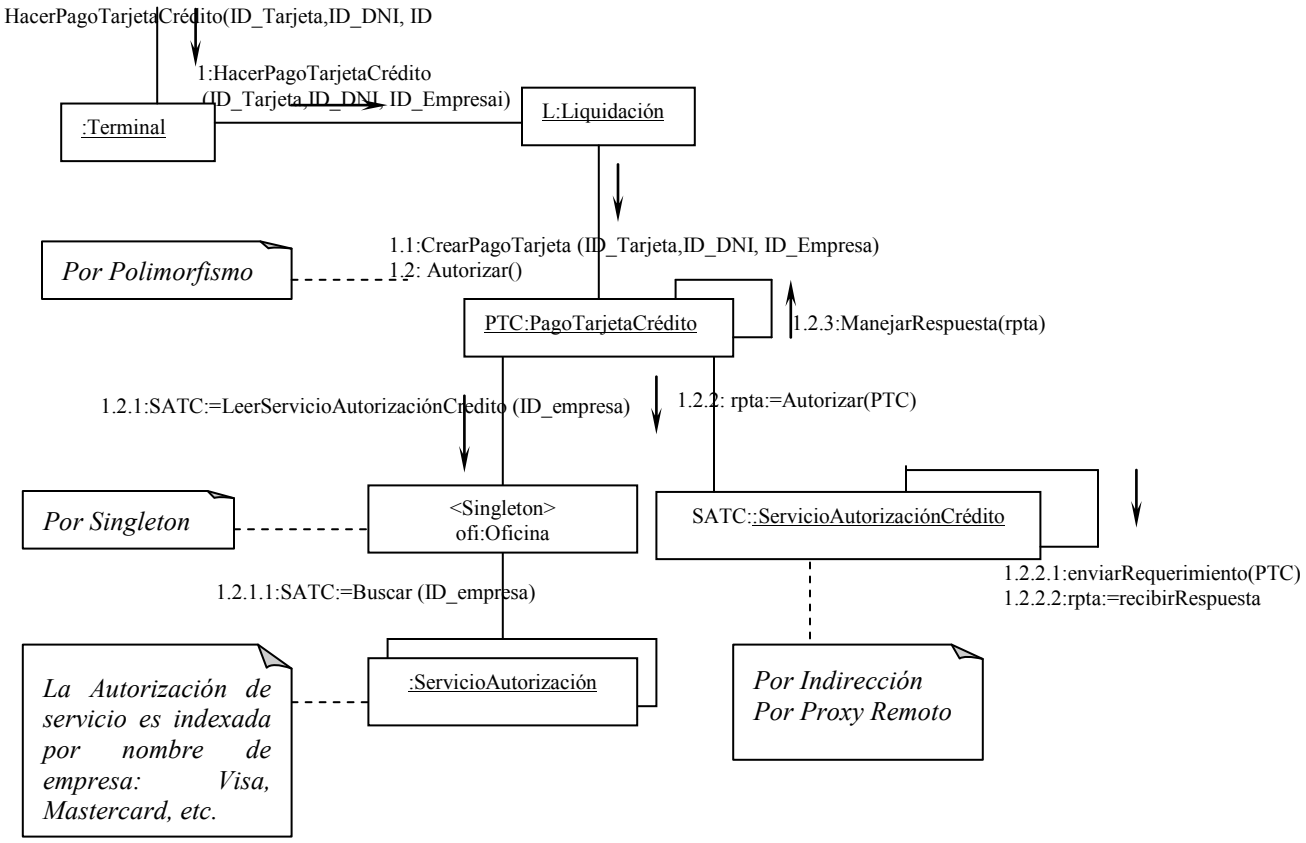


Figura 15.14
Diagrama de colaboración de la operación de sistema HacerTarjetaCrédito

Nota: *Como ya mencionamos anteriormente, el PagoTarjetaCrédito requiere asociarse a la Oficina para que devuelva al servicio de autorización basado en el nombre de la empresa. Sin embargo, la instancia de PagoTarjetaCrédito no tiene visibilidad sobre la Oficina. Una opción posible, puede ser que la Oficina sea pasada a través de la Terminal a la Liquidación y de la Liquidación al PagoTarjetaCrédito vía parámetro. Sin embargo, dado que Oficina es una clase con exactamente una instancia, creemos que es deseable que soporte visibilidad global. El patrón Singleton de GoF, sugiere la definición de un método que devuelva tal instancia. Luego PagoTarjetaCrédito tranquilamente puede comunicarse con la clase Singleton Oficina enviándole el mensaje que él necesite.*

15.8- Creación del diagrama de colaboración HacerPagoTarjetaBanco.

Estudiando el contrato de esta operación de sistema y el modelo conceptual de la aplicación, observamos que el diagrama de colaboración es bastante similar al anterior.

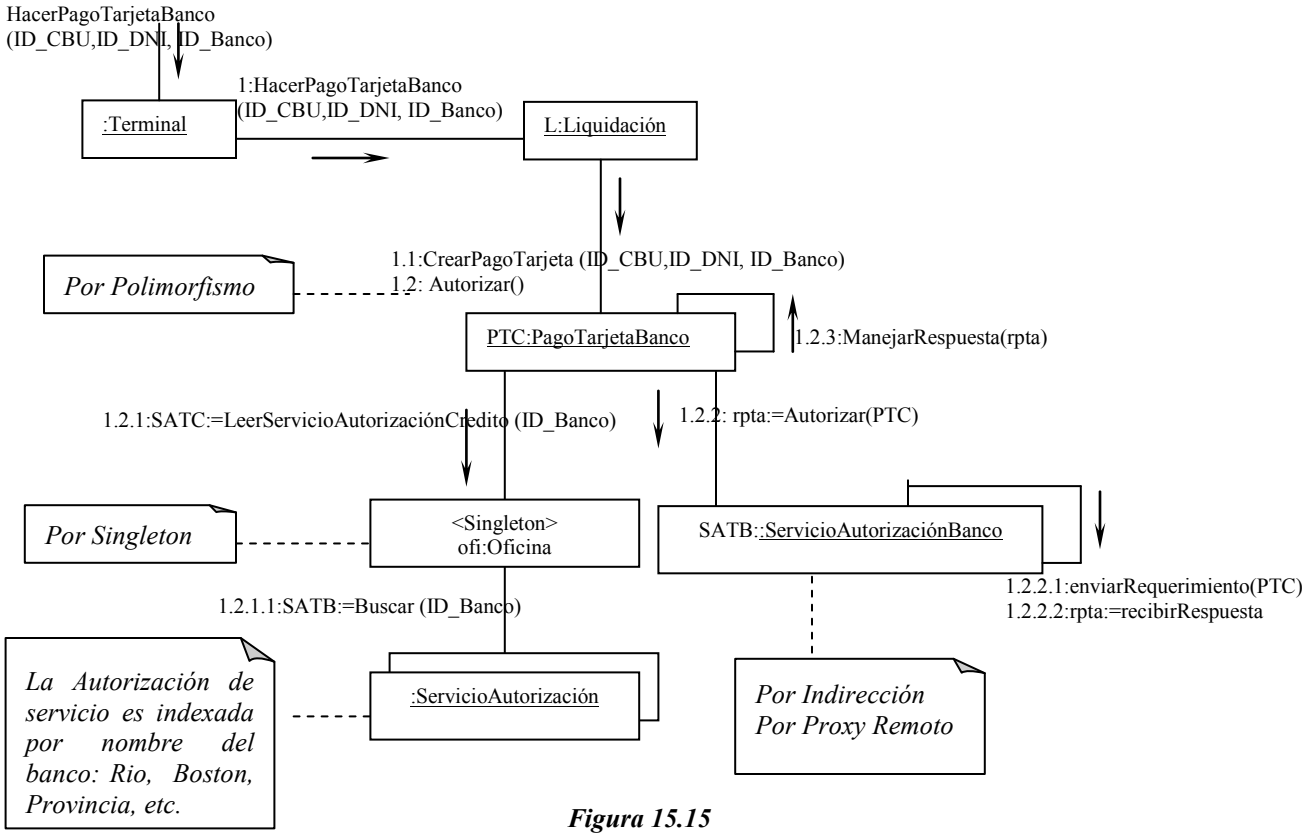


Figura 15.15
Diagrama de colaboración de la operación de sistema HacerTarjetaBanco

La secuencia sería:

- *Terminal le solicita a Liquidación que realice un pago por tarjeta bancaria.*
- *Liquidación crea una nueva instancia de PagoTarjetaBanco mandándole además los parámetros de inicialización.*
- *Liquidación le solicita al pago recién creado que se autorice (el PagoTarjetaBanco tendrá tal responsabilidad porque por experto es la clase indicada para realizar tal tarea por poseer toda la información que necesita).*
- *El PagoTarjetaBanco autoriza el pago enviándolo al Servicio de Autorización que obtiene de la oficina (que es Singleton).*
- *Cuando el servicio de autorización responde enviando la respuesta, PagoTarjetaBanco maneja dicha respuesta de manera apropiada.*

Nota: *Vemos que Servicio Autorización de Banco posee un mensaje polimórfico llamado "Autorizar". En el diagrama de clase se verá de la siguiente forma:*

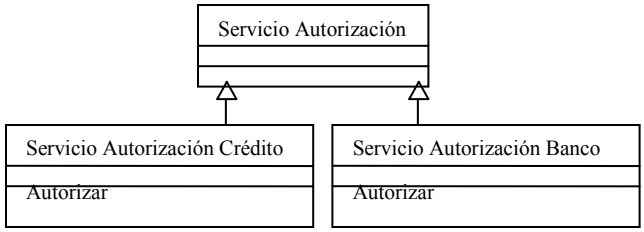


Figura 15.13

Capítulo 16 - Diagrama de Clases

16.1- Acerca de la organización del capítulo.

En este capítulo realizaremos las modificaciones pertinentes al diagrama de clase desarrollado en el primer ciclo de vida (capítulo 10). Para ello utilizaremos el modelo conceptual y los diagramas de colaboración desarrollados en los capítulos 13 y 15 respectivamente.

Seguiremos los pasos

- *Identificaremos todas las clases participantes en la solución de software.*
- *Separaremos las clases por paquetes.*
- *Esquematización del diagrama de clases: se incorporan a las clases los atributos y métodos y se incorporan las asociaciones entre las diferentes clases.*

Nota:

No sumaremos la información de tipos a los atributos y métodos para no dificultar la comprensión de los mismos.

Debido al tamaño del diagrama lo dividimos en paquetes según el diagrama del modelo conceptual.

16.2- Identificación de las clases de Software.

1- Existentes en el diagrama del primer ciclo

Terminal	*Oficina*
GeneradorBarra	*Liquidación*
Vehículo	*PagoBatch*
CatálogoVehículo	*CatálogoLiquidación*
CatálogoPagoBatch	*LíneaPeríodoLiquidado*
LíneaPeríodoVehículo	*PeríodoPagoBatch*

2- Nuevas clases

HojaLiquidación	*ServicioAutorización*
ServicioAutorizaciónTarjetaBanco	*ServicioAutorizaciónTarjetaCrédito*
PagoOnline	*PagoTarjetaBanco*
PagoTarjetaCrédito	

16.3- Enumeración de los paquetes y las clases asociadas.

Durante la realización del modelo conceptual, hemos dividido el diagrama en paquetes, para mejorar su compresión. Respetando esta separación a continuación esquematizaremos los paquetes y sus clases asociadas.

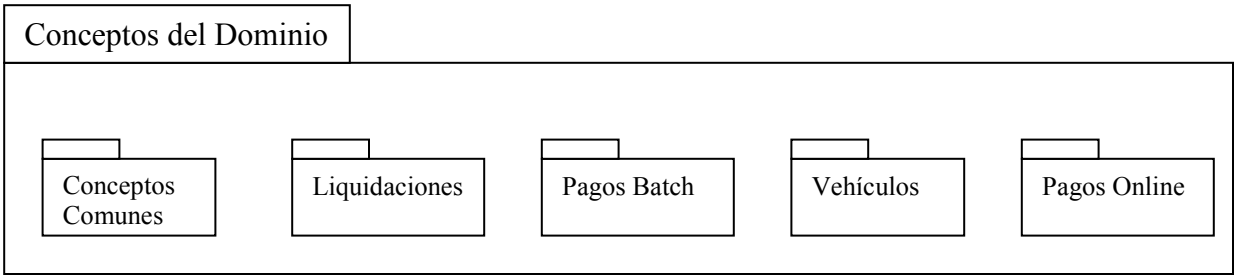


Figura 16.1
Paquetes incluidos en paquete "Conceptos del dominio".

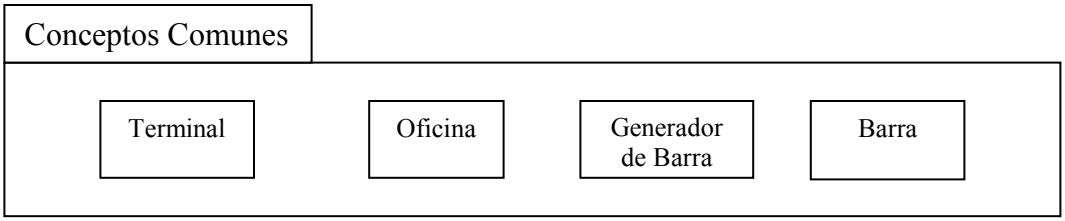


Figura 16.2
Clases incluidas en el paquete "Conceptos Comunes".

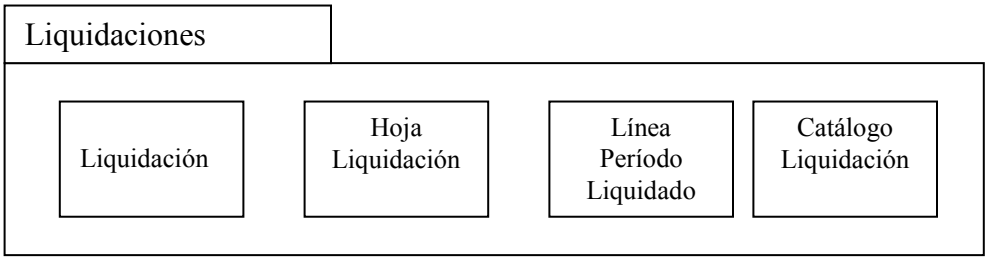


Figura 16.3
Clases incluidas en el paquete "Liquidaciones"

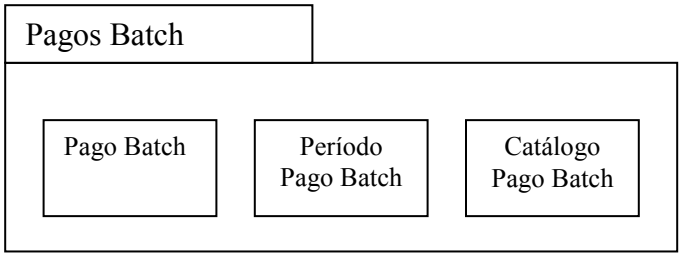


Figura 16.4
Clases incluidas en el paquete "Pagos Batch"

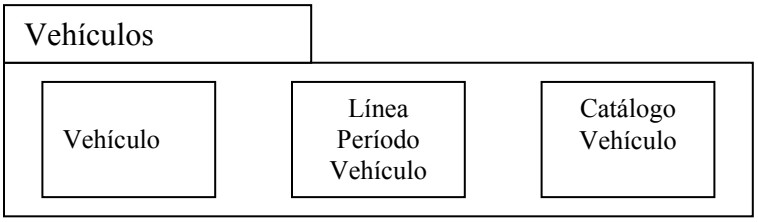


Figura 16.5
Clases incluidas en el paquete "Vehículos"

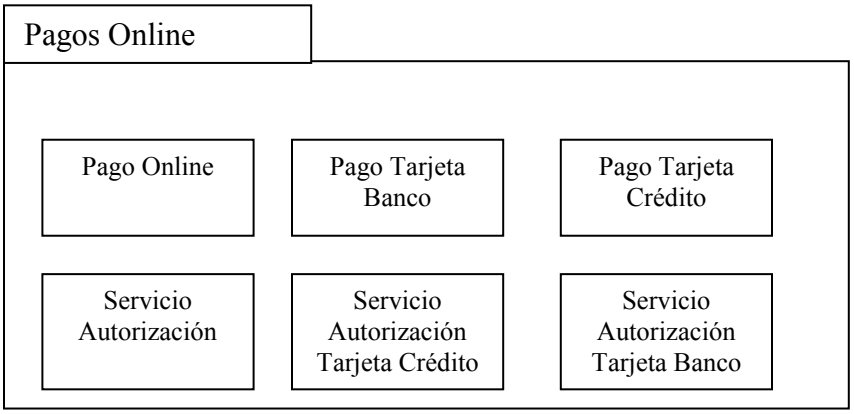


Figura 16.6
Clases incluidas en el paquete "Pagos Online"

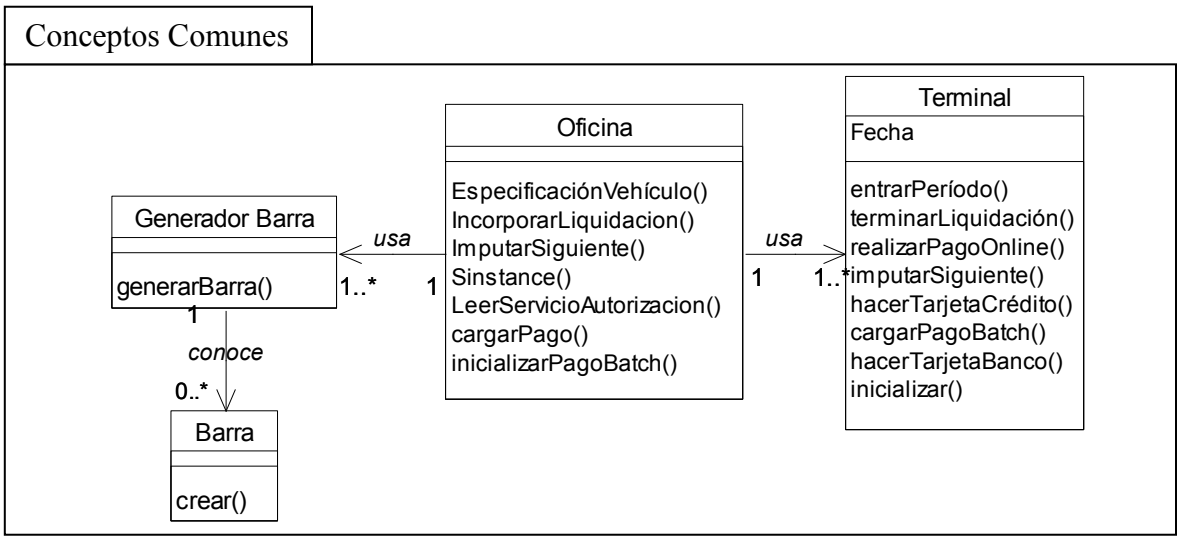
16.4- Diagrama de Clases

En las secciones subsiguientes esquematizaremos el diagrama de clases de la aplicación. Debido al tamaño del mismo, lo hemos dividido en paquetes según lo enunciado en la sección anterior.

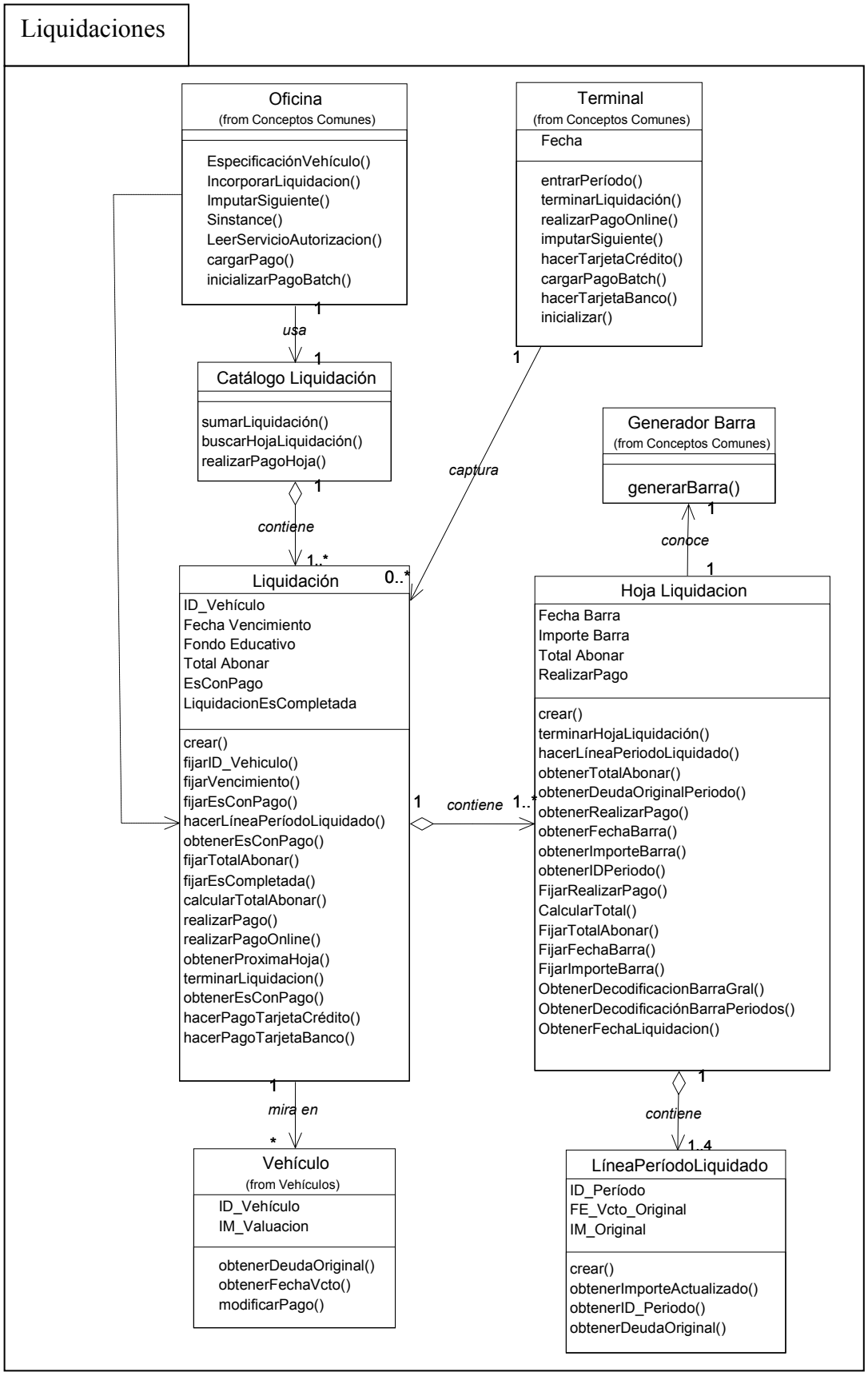
Tener en cuenta que:

- *Los atributos serán extraídos del diagrama conceptual.*
- *Los mensajes serán deducidos de la observación de los diagramas de colaboración. En general, el conjunto de todos los mensajes enviados a la clase X a través de todos los diagramas de colaboración indican la mayoría de los métodos que la clase X debe definirse.*
- *Las asociaciones serán deducidas del estudio de los diagramas de colaboración, en términos de que clase debe tener conocimiento de otra.*

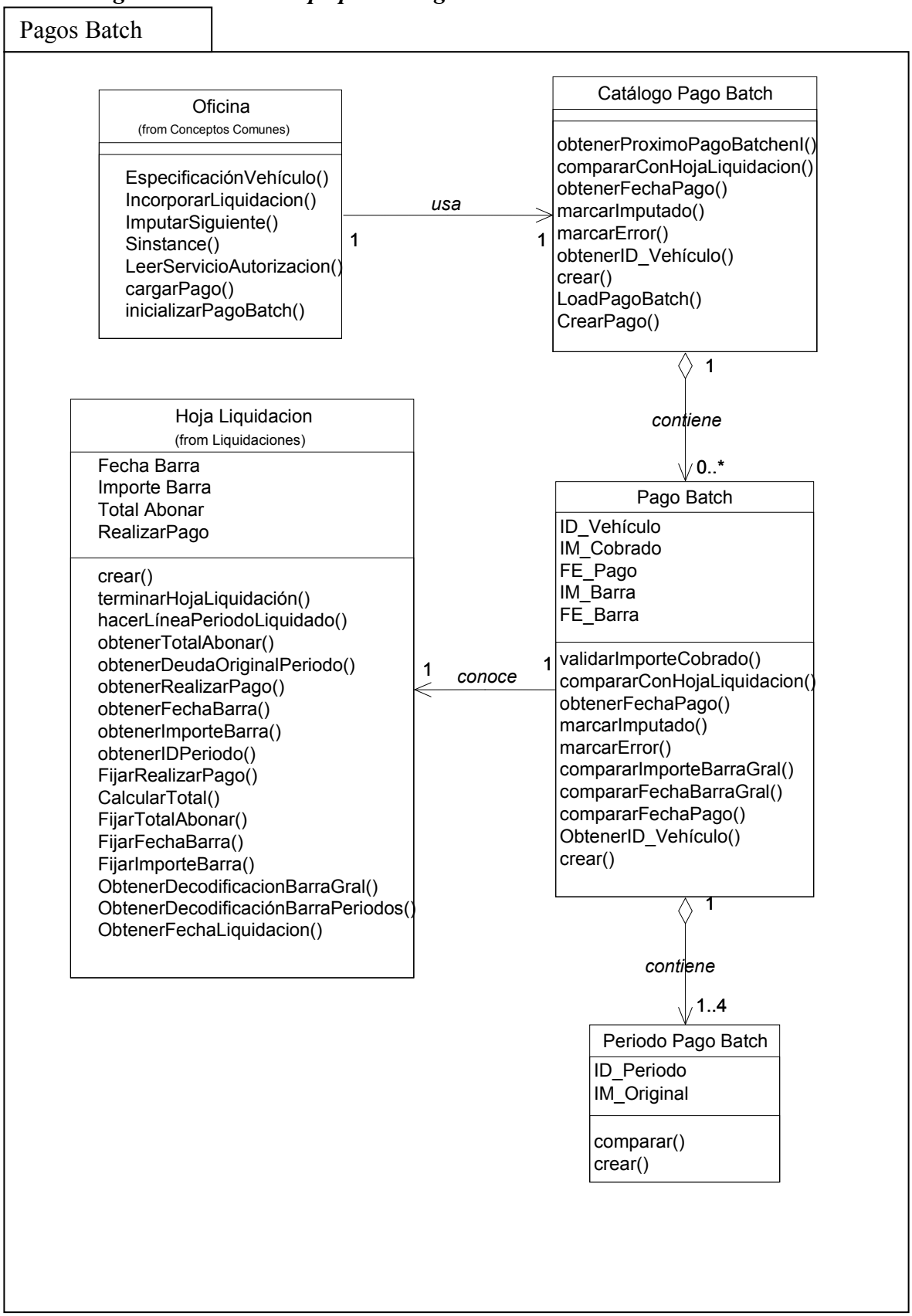
16.4.1- Diagrama de clase del paquete "Conceptos comunes".



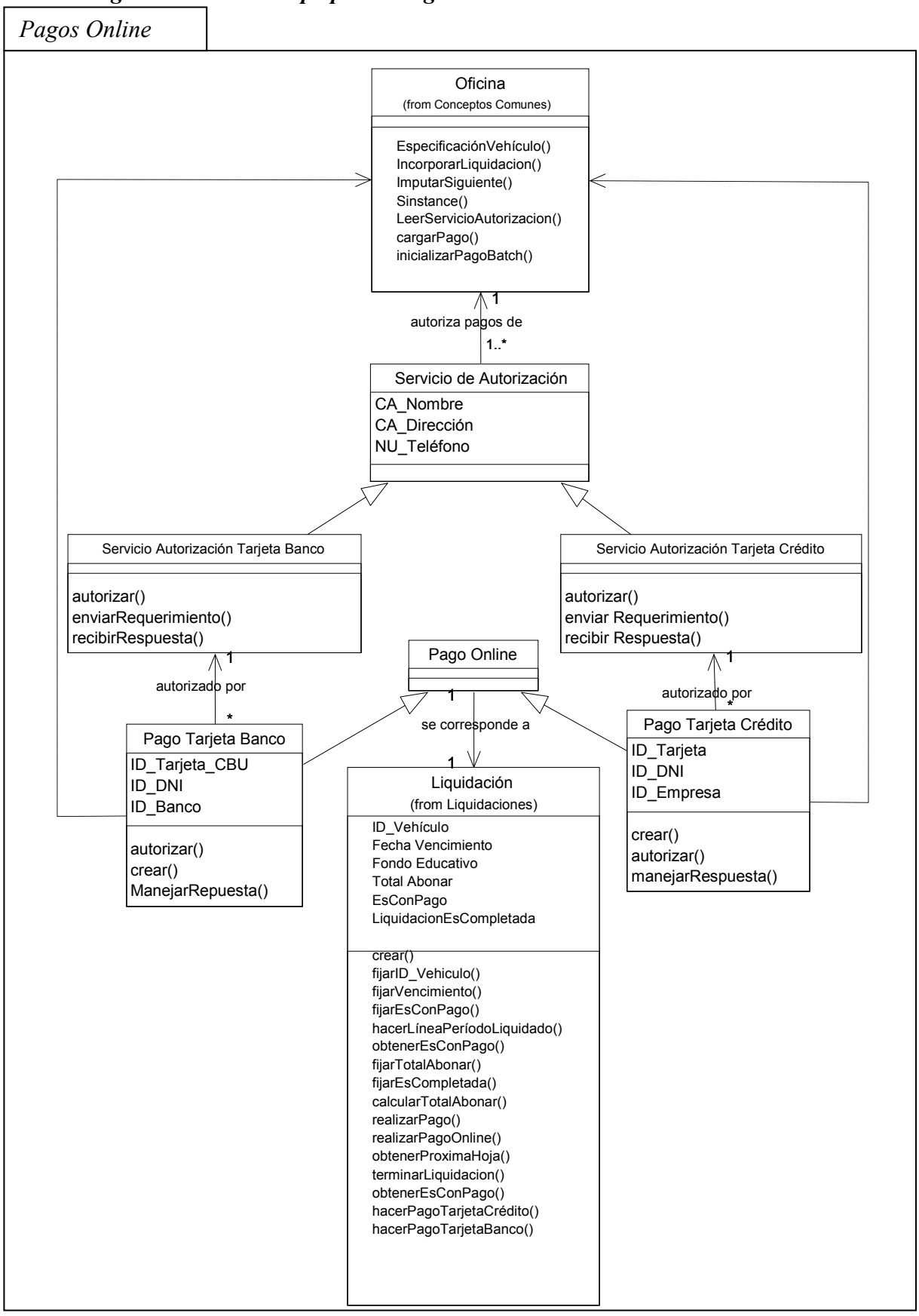
16.4.2- Diagrama de clase del paquete "Liquidaciones".



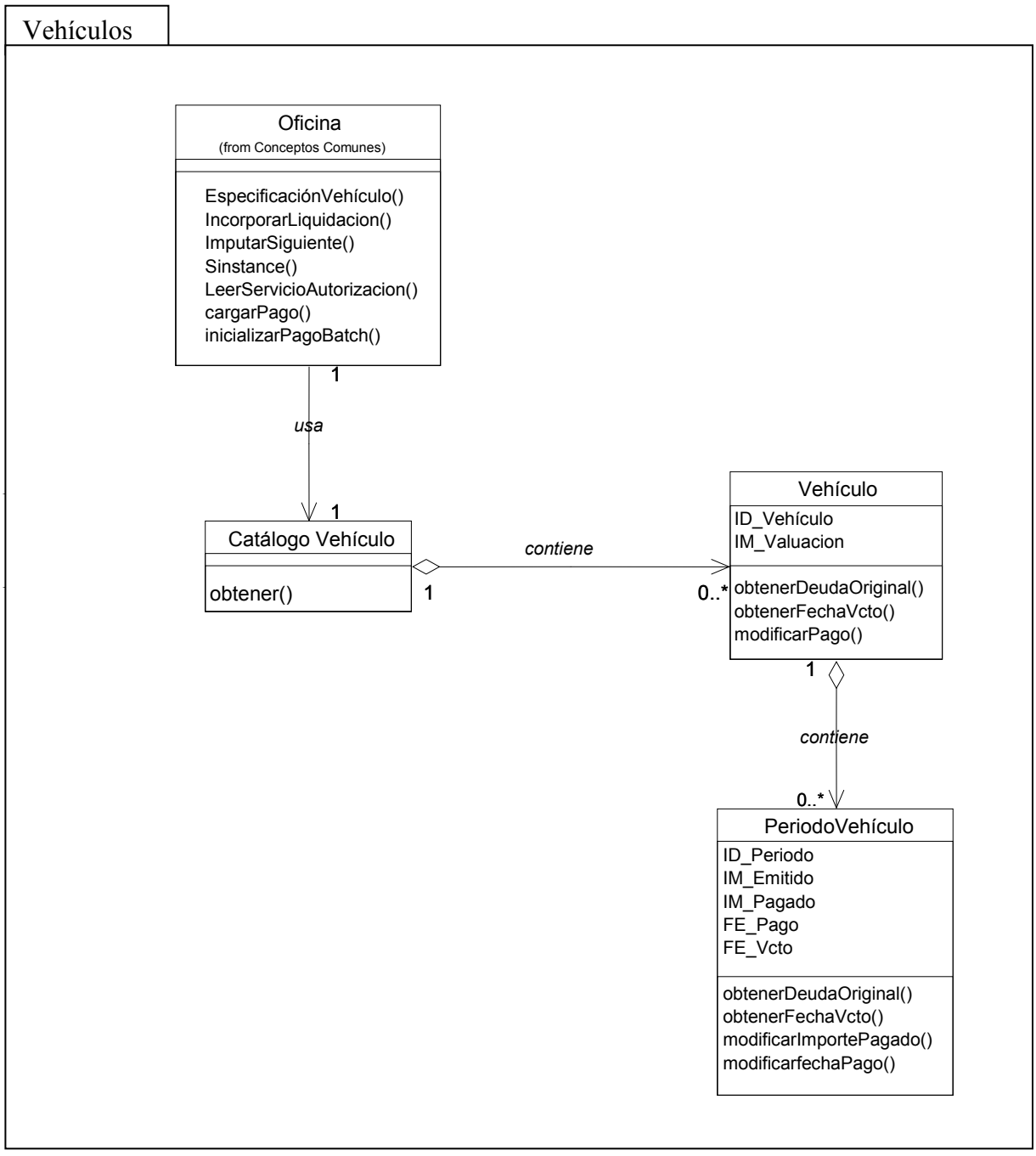
16.4.3- Diagrama de clase del paquete "Pagos Batch".



16.4.4- Diagrama de clase del paquete "Pagos Online".



16.4.5- Diagrama de clase del paquete "Vehículos".



Capítulo 17- Comparación de Metodologías de Trabajo

17.1- Introducción

El sistema que estuvimos desarrollando en los capítulos anteriores, ya se encuentra en producción en la actualidad. Sus desarrolladores utilizaron otra metodología para realizarlo. La idea de esta sección es encontrar los pro del proceso unificado de desarrollo de software y el paradigma orientado a objetos comparando la aplicación existente con la que desarrollamos en esta tesis.

Para que el lector logre una mayor compresión de tema, creemos conveniente describir primero la estructura del centro de cómputos donde se desarrolló el sistema de nuestro ejemplo, y la forma de desarrollo de una aplicación en el mismo.

Luego procederemos a la comparación de ambas aplicaciones.

17.2- Estructura del Centro de Cómputos

El centro de cómputos se encuentra dividido en varios sectores:

- ***Dirección***
- ***Seguridad:*** *encargado de proveer los accesos a los usuarios de las aplicaciones (Por ejemplo: un usuario puede tener acceso al programa de alta y de consulta y otro tener acceso a la consulta solamente)*
- ***DBA*** *(acceso a las bases de datos) encargado de:*
  - *La creación y mantenimiento de las bases de datos y de los files que contiene.*
  - *La creación, modificación y borrado de estructuras de files que se encuentran en bases de datos.*
  - *La creación de claves en los diferentes files: descriptores (definir un campo como clave), superdescriptores (definir varios campos como una clave) y subdescriptores (definir una parte de un campo como clave).*
  - *la creación de diferentes visiones de un file, etc.*
- ***Soporte técnico:*** *encargado de, entre otras cosas, la instalación de nuevos productos o versiones de Software, estudio de los productos de software (de los parámetros de inicialización de cada producto, de los parámetros ideales de inicialización del software del equipamiento central, etc.), realizar el soporte técnico del equipamiento central y de PCs.*
- ***TP:*** *encargado del Teleprocesamiento de los datos, de establecer el enlace con todas las terminales conectadas al centro de cómputos.*
- ***GraboVerificación:*** *encargado del ingreso manual de los datos necesarios para determinadas aplicaciones (cuando el ingreso de datos a un sistema es masivo).*
- ***Implementación:*** *encargado de la realización de procesos batch (ejemplo Imputación de pagos del Impuesto automotor)*

- ***Operaciones:*** *encargado del manejo de la consola (liberación de jobs, mandado a imprimir de formularios), manejo de dispositivos externos: Cartuchos, cintas, impresoras,etc.*
- ***Pyco:*** *sector de coordinación de las tareas batch del centro de cómputos (establece el orden y la prioridad de las mismas)*
- ***Sistemas:*** *encargado de desarrollo de aplicaciones (análisis y programación de las mismas)*
- ***Auditoria:*** *controla la forma en que los usuarios de las distintas aplicaciones utilizan el sistema (un mal uso de sistema puede acarear la eliminación de deuda de un contribuyente).*
- ***Mesa de ayuda:*** *constituye un centro de atención al usuario. El operador del sistema llama a este sector en caso de que se presentara una duda o considere que la aplicación tenga un mal funcionamiento. En caso de que los integrantes de este sector consideren que efectivamente existe un error en la aplicación, el mismo será reportado al sector de sistemas, para su solución.*

17.3- Metodología utilizada para el desarrollo de sistemas en el Centro de Cómputos

El sector de Sistemas es como ya lo dijimos el sector encargado de la realización de aplicaciones.

Internamente esta dividido en 2 grupos: los analistas y los programadores.

La forma de realización de una nueva aplicación es la siguiente:

- *Los analistas realizan un estudio de la aplicación a realizar mediante varias charlas con el/los usuarios del sistema, o sea realiza el análisis de requerimientos del sistema.*
- *En base a esas charlas surge un diagrama global del sistema como el que se muestra en el Anexo "Jerarquía del Sistema Online" en el que se dividen al sistema por funciones. Notemos que los analistas han utilizado el paradigma algorítmico.*
- *Con este diagrama en mano, los analistas crean las distintas estructuras de datos y las pasan al sector DBA para que las genere.*
- *Luego los analistas proceden a escribir los análisis de cada uno de los módulos que aparecen en la jerarquía del diagrama global (teniendo en cuenta la estructura de datos definida previamente)*

  La pregunta es ¿en qué consiste el análisis?

  Consiste en un documento con las especificaciones necesarias para que un programa realice las funciones deseadas. Estos escritos suelen tener muchas hojas dependiendo del tamaño del módulo a desarrollar. Un ejemplo de análisis es el que mostramos en el Anexo "Muestra del análisis de la Liquidación".
- *El análisis es entregado al programador, quien después de leerlo realiza las consultas pertinentes a los analistas. Muchas veces nos encontramos con modificaciones introducidas por el programador, ya que él posee una visión del lenguaje de programación que no tienen los analistas (ejemplo puede sugerir la creación de una clave para facilitar la escritura del programa).*

- *El programador procede a la codificación del análisis en sentencias entendibles por la máquina.*
- *El programador realiza la prueba del programa.*
- *Cuando considera que funciona, le avisa a los analistas para que procedan a realizar sus pruebas.*
- *Los analistas prueban el sistema y reportan los errores al programador quien va depurando los programas hasta lograr el correcto funcionamiento. Durante la realización de estas pruebas pueden darse cuenta de casos que no ha tenido en cuenta en su análisis, los mismos también son notificados al programador, quien procede a su incorporación en el o los programas que correspondan.*
- *Cuando consideran que el programa funciona, los analistas concertan una entrevista con el usuario para que vea el resultado obtenido. Durante esta charla muestran el nuevo sistema y entregan el manual de usuario (describe la forma de operar del sistema).*
- *Después los programas son pasados a un ambiente intermedio donde el/los usuarios pueden operar la aplicación antes de pasarlos al ambiente real. Frecuentemente se requieren modificaciones que son reportadas. Después de ser estudiadas, los análisis o bien le notifica al programador para que las codifique, o bien plantea al usuario una solución alternativa.*
- *Cuando el cliente está conforme con el producto obtenido, los analistas lo consultan acerca de los accesos que tendrán los diferentes usuarios a los distintos módulos de la aplicación.*
- *Luego se procede al pasaje de los módulos del sistema a ambiente real, y a informar los accesos al sector de Seguridad; quien realiza las acciones pertinentes para incorporarlos.*

17.4- Ambientes de trabajo

Existen 3 ambientes de trabajo, en cada uno de ello existe una copia de todos los módulos que constituye una aplicación:

1- ***Ambiente de desarrollo:*** *lugar donde los programadores codifican los módulos de la aplicación y donde se prueban los programas en desarrollo. Los datos de este ambiente, constituyen los lotes de prueba que arman el equipo de trabajo.*
2- ***Ambiente de testing:*** *Después que los analistas muestran el funcionamiento del sistema, todos los módulos de la aplicación mostrada son pasados a este ambiente, para que el usuario procesa a la prueba y a la familiarización del mismo. En este lugar los datos son una copia de los reales al lunes de cada semana.*
3- ***Ambiente Real:*** *Ambiente de trabajo de los usuarios.*

17.5- Adquisición de conocimiento de la aplicación.

17.5.1- Adquisición de conocimientos de la aplicación en el centro de cómputos.

Debido a la forma de desarrollo del sistema es muy difícil para un analista nuevo, adquirir rápido conocimiento de la aplicación.

Por un lado, tenemos ***el diagrama jerarquías de módulos****: es una herramienta realizada por los analistas que identifica muy generalmente las funciones de la aplicación (ver en el anexo "Jerarquía del Sistema Online"). Es una herramienta buena pero muy reducida.*

Este esquema no muestran los niveles inferiores de la jerarquía. Dichos niveles son considerados parte de la división interna de los programas que realizan los programadores. Estos niveles no son mostrados por 2 razones:

- *Estas divisiones son realizadas por los programadores en el momento de la codificación y por ende, los analistas no poseen conocimiento sobre ellos.*
- *Si se volcasen todas los módulos en el diagrama se volvería engorroso.*

Por el otro, tenemos ***el análisis de cada programa****: Si miramos el anexo "Muestra del Análisis de la Liquidación", veremos que es demasiado detallado y extenso. Los análisis son concebidos como un medio de comunicación entre los analistas y el programador, no como una herramienta para adquirir conocimientos de la aplicación.*

Debería de existir un diagrama intermedio, ni tan simplista, ni tan detallado.

Para que una persona pueda entender como funciona la aplicación en su totalidad es necesario, leer y entender todos los análisis escritos de la aplicación. Esto es sumamente dificultoso y consumirá mucho tiempo ya que (como se ve en el anexo "Muestra del Análisis de la Liquidación") al ser tan detallados se vuelven engorrosos y difíciles de seguir para alguien que no posee conocimiento previo. El nuevo analista, tendrá la difícil tarea de abstraerse de los detalles, para dilucidar, durante la lectura del análisis, las funciones del módulo que está examinando. Por otro lado, esta lectura no nos garantiza la comprensión cabal de la aplicación ya que no existe en ningún lugar una documentación con las asociaciones entre las diferentes partes de la misma.

17.5.2- Adquisición de conocimientos de la aplicación desarrollado por el proceso unificado.

Los diferentes modelos del proceso unificados (diagramas de casos de uso, casos de uso, diagrama conceptual, diagrama de secuencia, etc.) tienen las siguientes características:

- ***No poseen detalles de implementación*** *que dificultan la comprensión del sistema (si miramos el análisis del anexo "Muestra del Análisis de la Liquidación", notaremos que tiene formato de seudocódigo, nombrando hasta los nombres de los campos de los files).*

- ***Los diagramas son más legibles:*** *poseen un tamaño menor (en cuanto a volumen de hojas) al análisis anterior y muchas veces se encuentran representados mediante dibujos.*
- ***Los diagramas ofrecen una lectura rápida y concisa:*** *así tenemos los diagramas de caso de uso (que se ve a sola vista todas las funciones de la aplicación), la descripción de los casos de uso (que explica en una o 2 líneas que realiza cada módulo), el modelo conceptual (que muestra los conceptos de toda la aplicación y como se relacionan entre sí). Mirando todos estos modelos podemos saber que hace la aplicación de una manera rápida, y concisa.*
- ***Poseen una esquematización fácil de comprender:*** *los dibujos son más comprensibles. Generalmente para entender un análisis escrito debemos leerlo varias veces en su totalidad.*
- ***Muestran una visión global de la aplicación:*** *los diversos diagramas que se desarrollaron en esta tesis nos ayudan a comprender el funcionamiento global de la aplicación, ya que muestran las asociaciones entre los diferentes conceptos (ver ejemplo de sección subsiguiente).*
- ***Los diagramas son incrementales:*** *si se leen secuencialmente, se adquirirá conocimiento paulatino e incremental de la aplicación. Es más difícil que un nuevo analista se abrume por la cantidad de información que está adquiriendo.*

17.5.3- Ejemplo de Comprensión de Liquidación del impuesto.

El análisis de la Liquidación del impuesto en el centro de cómputos está compuesto de 25 hojas escritas (parte se muestra en el anexo "Muestra del Análisis de la Liquidación").

Para lograr comprender el funcionamiento del mismo es necesario:

- *Tener conocimiento de la aplicación o de alguien que explique.*
- *Disponer de un tiempo considerable para leer y comprender cabalmente de que se trata (a pesar del conocimiento previo que tuviese).*

Pero aunque leyeran las 25 hojas escritas y lograran comprender que es lo que realiza todo el análisis de la liquidación, no tendrán la visión global de la aplicación; ya que en este análisis no se muestra como se enganchan las diferentes piezas del sistema. Por ejemplo, decimos que hay que grabar la liquidación en el file de pago comprobado; pero no sabemos, a priori, para que lo hacemos, no sabemos que dicha hoja de liquidación será comparada en el momento de imputación con un pago, para ver si es correcto (para saber esto deberíamos leer las hojas y hojas de análisis del programa de imputación de pagos).

Vemos entonces que este análisis, a pesar de lo largo que es, no muestra a la aplicación en forma global sino en forma parcial. Si a esto sumamos los detalles de diseño que se encuentran dentro del mismo, podemos concluir que: para una persona que lo lee por primera vez y no tiene conocimiento de la aplicación resulta engorroso, ya que no puede dilucidar a primera vista cual es el funcionamiento de la liquidación y como se asocian sus conceptos con otros conceptos de la aplicación. Así, si un nuevo analista quisiera conocer toda la aplicación, deberá leer todos los análisis de la misma y tratar de

hacer las asociaciones de los distintos programas en su cabeza, cosa que le llevará meses sino años.

En el proceso unificado, en cambio, basta leer el diagrama de caso de uso Imputar Pagos para darnos cuenta de existe una asociación entre la hoja de liquidación y el Pago Batch.

En el curso típico de eventos, que mostramos abajo, remarcamos la frase que lo indica.

1-Este caso de uso comienza cuando el implementador dá la orden de que se proceda a la imputación de los pagos.	*2- Se tomarán todos los pagos cuyo estado de pago indique que se encuentra pendientes de imputación.* *3- Se realiza la validación de pago:* *3.1- Reconstruyendo el supuesto cobrado con el coeficiente de actualización (desde la fecha de barra a la fecha de pago) y el importe de la barra y verificando su igualdad con el importe cobrado en los datos del pago.* ***3.2- Obteniendo la hoja de liquidación asociada al pago ingresado (se encuentran en el catálogo de liquidaciones). Esta liquidación posee las siguientes características:*** ***Continua el proceso***

17.6- Redundancia.

Estudiaremos 3 aspectos que influyen en la redundancia de la codificación en el centro de cómputos:

- *Las visiones del analista y del programador sobre la aplicación.*
- *La forma de localizar los módulos ya programados.*
- *La ausencia de la característica de herencia existente en el paradigma orientado a objetos.*

17.6.1- Visión del analista y del programador sobre la aplicación.

El paradigma algorítmico supone la división del sistema por funcionalidades o módulos. Esta división la realizaron los analistas del sistema que funciona en la actualidad. Ellos son las personas que hicieron el análisis de requerimientos y que poseen, por ende, la visión global de la aplicación. También escribieron el análisis de cada módulo y lo entregaron a los programadores quienes lo codificaron en sentencias entendibles por la máquina.

La aplicación tiene más de ***un programador*** *y los mismos* ***carecen del conocimiento global****, ellos se dedican a transformar los análisis que les dan en programas eficientes. Cuando dicimos eficiente, nos estamos refiriendo a:*

- ***Tiempo que tarda la ejecución del programa:*** *cuando estamos ante un programa que puede tardar mucho tiempo de máquina (generalmente programas batch) debido al gran volumen de información que está procesando (por ejemplo: el cálculo del impuesto a los automotores implica el cálculo de las 3 cuotas que deben pagarse en el año para aproximadamente 4.5000.000 de dominios), la forma de realizar una lectura o de actualizar una base se vuelve esencial. Una mala organización del programa puede llevar a que se tarde 30 horas o más, en vez de 6 horas (por ejemplo).*
- ***La forma de organizar y dividir el programa:*** *se trata de organizar un programa de manera que sea fácilmente legible. No siempre es el mismo programador el que va a modificarlo en el futuro y por ende, se piensa en eso a la hora de la realización. Los comentarios, la forma de nombrar a las variables, y la división en subrutinas o subprogramas cuando la lógica se vuelve difícil de leer, son importantes.*
- ***La prueba del módulo una vez terminado:*** *es importante que el programador realice las pruebas de su programa antes de la prueba realizada por los analistas, para detectar los errores más grandes (como que no chequee o que no haga lo que dice el análisis)*

Como se ve, la tarea del programador no es ver la aplicación en su totalidad, sino realizar lo más eficientemente posible los módulos que los analistas le entregan para que codifique. Esto hace a la redundancia en los programas, pues distintos módulos pueden poseer partes repetidas.

Por ejemplo:

- *El programa de alta de la aplicación dice que la fecha de vencimiento de la cuota del alta es 15 días hábiles desde la fecha de vigencia del alta.*
- *El programa de liquidación por el otro lado, dice que la fecha de vencimiento de la liquidación son 5 días hábiles desde la fecha corriente.*
- *Claramente vemos que podría haber un módulo que se corriera n días hábiles a partir de una fecha estipulada por parámetro. Sin embargo como dichos programas fueron realizados por distintos programadores que no conocían (o por lo menos no al detalle) lo que el otro estaba implementando, resultó en que ambos codificaron lo mismo 2 veces dentro de cada módulo.*
- *Por el otro lado, los analistas no pudieron darse cuenta de la duplicidad ya que ellos no estructuran el programa, y esto hace a su forma de organización interna. Ellos dividieron la aplicación en grandes módulos tales como Alta, Baja, Liquidación, etc. y dieron cada función a un programador para que la realizara.*

En el paradigma orientado a objetos, no ocurre esto ya que lo que codifica el programador son los mensajes de los objetos. Lo único que puede estar repetido, es las funciones dentro del mismo objeto, pero en todos casos se encuentra más aislado y es más fácil de ubicar esa redundancia de código. El programador antes de codificar cualquier

mensaje se fijará si ya existe en la clase en que debe codificarlo. En nuestro ejemplo, seguramente existirá una clase Fecha en la que se colocarán todas las funciones relacionadas a la fecha. Cuando el programador necesite calcular la fecha de vencimiento de la cuota del alta, naturalmente buscará en la clase Fecha si existe una función que lo ayude a resolver su problema, si no existiese la codificará lo más general posible, estando entonces disponible para futuros programadores (por ejemplo el que codifica el programa de Liquidación).

17.6.2- Forma de localizar los módulos ya programados.

Cuando el programador codifica un nuevo programa, utiliza en el proceso de creación varios objetos de diferente tipos: mapas (son las pantallas de la aplicación), subprogramas (programas que deben ser siempre llamados desde uno principal), locales (áreas donde se ubican la definición de variables locales al objeto en desarrollo), globales (área donde se ubica la definición de variables globales a todos los módulos del programa). Los nombres de los módulos tienen la restricción de que deben ocupar 8 carácter. En el centro de cómputos, los 3 primeros identifican a la aplicación, el cuarto carácter identifica el tipo de objeto que se trata (ejemplo N es subprograma, M es un mapa, G es una global, etc.) y por último, de la quinta a la octava posición identificamos la función del módulo.

En definitiva tenemos 4 caracteres para identificar la funcionalidad del módulo. Esto provoca la creación de programas con nombres no muy legibles, lo cual no favorece a la ubicación del lugar donde puede encontrarse codificada determinada función. Muchas veces, el programador codifica algo que sabe que ya hizo antes, pero que no recuerda que nombre de módulo tiene.

En cambio en la metodología orientada a objetos, podemos deducir más naturalmente el lugar donde se encuentran las cosas, ya que las clases poseen nombres relacionados al concepto que ellas representan y poniéndonos a pensar que clase tiene la responsabilidad de realizar la función buscada, es factible ubicar el lugar. En el ejemplo de la sección 17.6.1, si estamos buscando una función que se mueva n días hábiles y existe una clase Fecha, entonces es razonable pensar que ella es la que posee el método que devuelve la fecha deseada.

17.6.3- La ausencia de la característica de herencia.

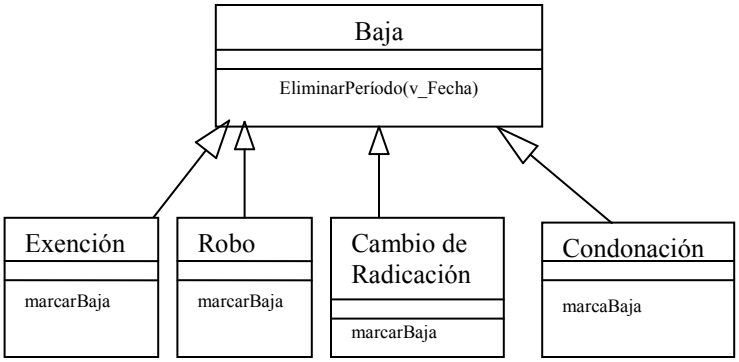
La herencia es una característica propia del paradigma orientado a objetos. Mediante este concepto un subtipo hereda todos los atributos y funciones de su supertipo y agrega propios. Esto ayuda a evitar la redundancia en la codificación de los programas.

Consideremos el ejemplo de las funciones de Baja (un vehículo no paga más el impuesto a partir de determinada fecha):

El sistema permite la baja de un dominio de la aplicación por varias razones:

- *El vehículo se encuentra exento de pago (es una ambulancia, pertenece al poder judicial o ejecutivo, etc).*
- *El vehículo va a ser radicado en otra provincia.*
- *El vehículo fue robado.*
- *El vehículo fue condonado del pago.*

Estos módulos tienen muchas cosas en común. Utilizando herencia podríamos haber realizado un esquema de la siguiente forma:



Así las funciones comunes (tal la eliminación de los períodos del Vehículo) serán realizadas por la clase Baja (superclase) y las funciones distintivas (marcar el vehículo como que fue robado o exentado o que se fue de la provincia o fue condenado) serán realizadas por la subclase que corresponda (Exención, Robo, Cambio de Radicación o Condonación). De esta manera codificaremos las funciones comunes una sola vez(en la superclase) y no por cada tipo de baja.

17.7- Modificación del sistema

Los aspectos que trataremos con respecto a la modificación de la aplicación son los siguientes:

- *Documentación de las modificaciones.*
- *Estimación del impacto de las modificaciones.*
- *Impacto de una modificación.*
- *Facilidad de una modificación.*

17.7.1- Documentación de las modificaciones.

En el centro de cómputos, muchas veces ocurre que debido a la urgencia que tienen las modificaciones, las mismas no son volcados en el análisis (quedando así desactualizados). Esto provoca que muchos detalles conceptuales no queden documentados, solo queda en la mente de los analistas de turno. Además muchas veces esos conceptos no documentados son olvidados y resulta en la prueba del sistema para ver como funciona ante determinados casos.

Por ejemplo, hace un tiempo atrás los analistas, debían incorporar al sistema una nueva forma de descuento para los vehículos de transporte público, debido a que no recordaban todos los casos posibles de la rutina de cálculo del impuesto, luego de leer el análisis de la rutina y ver que estaba desactualizado, tuvieron que probar los casos faltantes generando lotes de prueba en el ambiente de desarrollo (ambiente de prueba del sistema).

La pregunta que nos surge es ¿Porqué no se documentan correctamente las modificaciones?

La respuesta es que existe una brecha muy grande entre el análisis y la programación. El análisis sirve para que sea leído por el programador. No existe una proximidad entre el lenguaje utilizado y el análisis, solo le dice al programador que hacer. Luego, cuando los tiempos apremian, frecuentemente los analistas, con trabajo por demás, expresan al programador verbalmente la modificación a realizar.

Para concluir, ***todo lo escrito por los analistas constituye una guía a seguir y una forma de mantener documentada la aplicación, pero no ayuda al programador en la tarea de codificar el programa. El módulo ejecutable debe ser pensado y codificado en su totalidad por el programador.***

En el proceso unificado, la realización de los diagramas ayuda a la hora de realizar la codificación. Para entender mejor el concepto basta con mirar el diagrama de clase obtenido en los capítulos 10 y 16, y ver que en el mismo los diseñadores especificaron todos los atributos de una clase y los mensajes a desarrollar. El programador toma el diagrama de clases y el diagrama de colaboración y los traduce: crea las clases con las variables de instancias y los métodos que se encuentran en el diagrama de clases, y codifica cada método siguiendo la secuencia de llamadas que especifica el diagrama de colaboración. Seguramente hasta es posible que los diseñadores creen las clases con sus variables de instancias y el programador solamente codifique métodos de las mismas.

17.7.2- Estimación de Impacto de las modificaciones.

El impacto de una modificación en el sistema, es más fácil de estimar cuando se utiliza el paradigma orientado a objetos. Debido al principio de encapsulamiento de las clases por el cual se ocultan las características internas de una clase dentro de la misma (la clase sólo muestra al exterior sus mensajes y los parámetros de los mismos), la evaluación de un cambio será más fácil, ya que implicará la modificación de las clases asociadas al mismo.

Además los modelos utilizados en el proceso unificado de desarrollo de software (tales como el conceptual, diagrama de colaboración o de clases) también ayudan a la estimación del impacto, debido a su naturaleza global.

Como contrapartida, en el sistema de centro de cómputos es difícil de estimar el impacto de una modificación; debido a que puede propagar a varios módulos, que no necesariamente tienen que ver con el cambio que estamos realizando. Luego se requiere de un seguimiento muy exhaustivo del sistema. No existe, en esta metodología de trabajo, un diagrama en el que, a priori, se pueda evaluar una modificación. Muchas veces esta carencia, provoca la prueba de toda la aplicación para realizar la estimación.

17.7.3- Impacto de las modificaciones

Consideremos, por ejemplo, que debemos modificar la estructura de datos:

1- En la ***aplicación que desarrollada en esta tesis:*** *la incorporación, modificación o borrado de un atributo de una clase afecta principalmente a la clase en sí (esto se debe principalmente a la característicade encapsulamiento de la metodología orientada a objetos). Luego si modificamos, borramos o incorporamos un atributo de una clase se deberá examinar minuciosamente todos sus métodos para evitar errores.*

2- En ***la aplicación desarrollada en el Centro de Cómputos****: la incorporación, borrado o modificación de campos (que se encuentran en files que son declarados en cada módulo que lo utiliza), implica:*

- El campo es modificado, o incorporado o borrado físicamente en el file.

- El campo es incorporado, borrado o modificado lógicamente en el file (se incorporan en las visiones del file que se requiera).

- Luego se deberán modificar todos los programas que hagan uso del campo.

En el sistema desarrollado en el centro de cómputos, los cambios estructurales tienden a impactar más profundamente en la aplicación debiendo modificar módulos muy diversos, a veces de gran tamaño y muy complicados.

En la aplicación desarrollada en esta tesis mediante el paradigma orientado a objetos, la propiedad de encapsular los datos dentro de clases, provoca que las modificaciones no se propaguen, en demasía, a lo largo de todo el sistema. *Así cuando tuvimos que incorporar la restricción de que en una hoja de liquidación podía haber a lo sumo 4 períodos, afectó a clases tales como Liquidación u Hoja de Liquidación (clases que tienen que ver con la modificación), pero no a otras tales como Pago Batch.*

A continuación, en las secciones subsiguientes, expondremos ejemplos asociados a estos conceptos

17.7.3.1- Ejemplo de modificación estructural: modificación del código de marca.

En la aplicación existe un campo que identifica la marca que posee un vehículo. Este campo, hasta hace poco, poseía 5 posiciones alfa – numéricas:

- *Las dos primeras identifican a la fábrica. Por ejemplo: RN es un Renault o PG es un Peugeot.*
- *Las tres últimas identifican la línea del vehículo. Por ejemplo el RN012 es un renault 12.*

Debido a que la fábrica de Ford tenía más de 999 vehículos de diferentes líneas, se hizo necesario el aumento del campo de código de marca de 5 posiciones a 7. Así el código de marca que antes era RN012 pasó a ser RN00012.

En el centro de cómputos requirió la modificación del programa de alta, modificación de datos, rectificación de datos de un vehículo. Luego se modificaron todas las pantallas del sistema donde se mostraba este campo quedando así afectados programas tales como baja de un vehículo por exención o robo, transferencia de un vehículo, consulta, etc. Como podemos observar, el impacto del cambio en el sistema fue grande y, por ende, debió ser globalmente probado.

En cambio, en el paradigma orientado a objetos, el código de marca sería un atributo de la clase Vehículo. Al estar contenida en dicha clase, deberemos estudiarla para ver si el cambio no afecta a los métodos de la misma. Así el lugar de modificación no se propaga y está más acotado a la clase que tiene el atributo que se cambia.

17.7.3.2- Ejemplo de Incorporación del concepto de liquidación de un período.

El concepto de liquidación de un período constituye dos caracteres que aparecen en cada línea de período liquidado (de una liquidación), que indica si un período es adeudado en su totalidad (SP), en forma parcial (DI), o está cancelado (CA) (muchas veces se solicita la liquidación de períodos cancelados, como comprobante de pago del impuesto).

En el caso del paradigma orientado a objetos la incorporación de esta funcionalidad implicará la creación de un nuevo atributo en la LíneaPeríodoLiquidado (concepto de pago) y de un método en la clase Vehículo que devuelva el concepto de liquidación y de otro método en la clase LíneaPeríodoVehiculo fije el concepto a cada una de las líneas. Se requerirán también retoques en las clases Liquidación, LíneaVehículo y Hoja de Liquidación para que pasen por parámetro el concepto a la clase Vehículo. Todas las demás clases (tales como: PagoBatch y Pago Online) no sufrirán cambio alguno. Todos los métodos a modificar o crear resultan pequeños, de hecho el más grande sería el que tiene la lógica de comparar el importe emitido y el pagado (en LíneaPeríodoVehículo) y en base a la comparación ver que concepto devuelve. El impacto en el sistema claramente se ve mirando el diagrama de clases.

El sistema ya desarrollado en el centro de cómputos, posee un programa de más de 2000 en donde se lista la liquidación, es él el que deberá absorber dicha incorporación, con todas las dificultades que acarea la prueba del mismo. Esta deducción se logra realizar después del estudio minucioso de la aplicación, puesto que no se tiene una herramienta que muestre globalmente a la misma.

17.7.4- Facilidad de una modificación.

Consideramos que el proceso unificado de desarrollo de software y el paradigma orientado a objetos hacen a la creación de una aplicación más fácil de modificar. Esto se debe a 3 factores:

- ***El proceso unificado fue concebido para crear aplicaciones cambiantes:*** *al realizar una aplicación en pequeños miniproyectos que incorporan nuevas funcionalidades, estamos constantemente modificando a la aplicación. De esta manera un cambio en el sistema, implica un ciclo (miniproyecto) más del proceso unificado. En cambio en la metodología del centro de cómputos, los sistemas fueron analizados, diseñados y luego implementados en una sola etapa. Una modificación, implicará una revisión de toda la aplicación.*
- ***El principio de encapsulamiento:*** *por este principio cuando modificamos la aplicación sólo tocamos las clases asociadas al cambio, evitando la propagación.*
- ***El proceso posee modelos que muestran globalmente la aplicación:*** *estos diagramas ayudan a la estimación del impacto de la modificación.*

17.8- Tamaño de los módulos.

En esta sección evaluamos el tamaño de un programa en términos de la cantidad de líneas codificadas.

Examinando los diferentes programas desarrollados en el centro de cómputos en términos de su tamaño, notamos que, si bien hay pequeños, muchos que son extensos (por ejemplo: el programa de Liquidación, el de Imputación o el de Rectificación).

En las secciones subsiguientes enunciaremos algunas desventajas de la creación de módulos muy voluminosos, para luego estudiar y ejemplificar las causas por las cuales consideramos el proceso del centro de cómputos provocan la creación de programas grandes.

17.8.1- Desventajas de la generación de programas de gran tamaños

- ***Su modificación es más difícil y riesgosa:*** *La modificación de programas grandes no es tarea sencilla. Es un trabajo costoso y riesgoso.*
  Si el programador no lo conoce y no logra dilucidar la lógica del programador original puede desvirtualizarlo, modificándolo de manera errónea, disminuyendo en forma notoria su tiempo de vida. Ejemplo: En el programa de liquidación como el programador no encontraba el lugar exacto donde se calculaba la fecha de vencimiento de la barra, resultó que cuando tuvo que modificar dicha fecha para determinada opción, la calculó como 3 veces dentro del programa, ya que en algún lado la perdía. En un programa pequeño dicha lógica es fácilmente deducible.
- ***Son poco cohesivos:*** *generalmente realizan tareas muy diversas y de distintos tipos.*
- ***Son más difíciles de probar:*** *requiere la generación de lotes de prueba extensos con gran variedad de casos y esto requiere más tiempo.*
- ***La detección de errores es más difícil:*** *poseemos más variedad de casos para la prueba, siendo más probable la omisión de alguno.*
- ***Mayor posibilidad de disminuir el ciclo de vida:*** *los programas grandes desde su nacimiento, no tienden a volverse chicos, más bien todo lo contrario es cierto. Dichos programas son cada vez más grandes, ya que, con el transcurrir del tiempo frecuentemente se le agregan más y más funcionalidad y, por ende crecen y crecen, hasta llegar el punto de que se deben reescribirse ya que son inmanejables e ilegibles.*

17.8.2- Razones por las que se crean programas de mayor tamaño.

Hemos detectado 3 razones que facilitan la creación de programas con mayor volumen de sentencias en la metodología utilizada en el centro de cómputos:

1-El paradigma algorítmico: *por la utilización del paradigma algorítmico, dividimos la aplicación en funcionalidades tales como alta, baja, etc. Estos módulos abarcan muchas tareas. De hecho los análisis de la aplicación, frecuentemente son extensos porque deben explicarlas. Luego, a la hora de codificar, los programas deben realizar las tareas que indican los análisis. Como resultado tendremos una alta probabilidad de obtener programas extensos.*

Creemos que la utilización del paradigma orientado a objetos, lleva a la realización de programas más pequeños (en cantidad de líneas programadas) con respecto a la otra metodología, debido a la división por objetos en vez de por funciones.
La modularización en objetos es más eficiente. Cada objeto realiza las tareas asociadas a los datos que él posee y en caso de requerir de información de otro objeto, se la solicitará al mismo (no la obtendrá él, como en el modelo algorítmico), reduciendo de esta manera el tamaño de la codificación.
Ejemplificaremos lo expuesto, mediante el ejemplo 17.8.3.

2- En la aplicación desarrollada en esta tesis, la asignación de responsabilidades a los objetos está justificada mediante la utilización patrones: *los patrones sugieren evitar la creación de objetos con demasiadas reponsabilidades ya que provocan clases extensas. A la hora de la creación de cada uno de los modelos, constantemente estamos pensando la asignación de responsabilidades, y utilizamos patrones (GRASP[Larman97] en nuestro caso) para que dicha asignación no se vuelva poco cohesiva o con alto acoplamiento. Es una asignación más justificada y pensada, siendo poco probable la generación de objetos muy extensos.*

3-La ausencia de las características de herencia y polimorfismo: *además de evitar la redundancia en la codificación, la herencia ayuda a reducir el tamaño de los programas. Ejemplificaremos este concepto en el ejemplo de la sección 17.8.4.*

17.8.3- Ejemplo: modularización del programa de Alta de un Vehículo.

En el centro de cómputos el análisis del programa de alta, muestra que debe realizar las siguientes funciones:

1- Mostrar las pantallas donde se ingresan los datos del vehículo y del contribuyente.
2- Validar los datos de las pantallas.
3- Controlar las teclas presionadas.
4- Actualizar todos los file asociados al alta de un dominio (file donde están los datos del vehículo, file donde están los períodos a tributar por el vehículo, etc.).

El conjunto de tareas enunciadas en los puntos 1, 2, 3 y 4 deben ser realizadas por un único módulo, que dependiendo del programador se divide en subprogramas o no.

En el paradigma orientado a objetos dividimos la aplicación en objetos que interactúan entre sí para lograr un objetivo. Así el alta de un vehículo implicaría:

1. *La creación de un objeto de la clase Vehículo: la clase Catálogo de Vehículo (por patrón creador) tendrá tal responsabilidad y para su creación deberá solicitar los datos de sus atributos a la Terminal. Por cada atributo de la clase Vehículo habrá un método que fije y otro que valide el valor del mismo. Cuando tenga que generar los períodos del Vehículo seguramente existirá otro método dentro de la clase que realice tal tarea teniendo en cuenta los atributos del vehículo (este método involucrará el envío de mensajes a la clase PeríodoVehículo para que inicialice y que cree nuevos instancias).*
2. *La incorporación del mismo en el Catálogo del Vehículo que se comunicará con un proxy quien será responsable de interactuar con la base de datos.*

Notemos que la metodología utilizada dividió al programa de alta en varios métodos en la clase Vehículo, Catálogo de Vehículo y PeríodoVehículo. Luego, se separó en módulos más pequeños independientes entre sí. La forma de modular el programa resultó más eficiente.

17.8.4- Ejemplo: Utilización de Herencia en el programa de Cese.

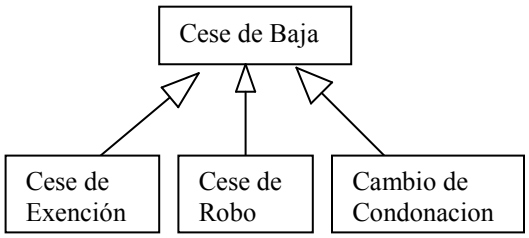
La funcionalidad de Cese de exención del impuesto implica que un vehículo a partir de determinada fecha comienza a tributar nuevamente el impuesto. Por ejemplo:

- *Si un vehículo fue robado y unos meses después es encontrado, entonces no debe pagar los períodos en donde estuvo robado, pero sí los períodos posteriores a la fecha en que se encontró. Así, el cese de un robo, provocará que un dominio comience nuevamente a pagar el impuesto desde la fecha en que se encontró el vehículo en adelante.*

Existen otras 2 causas por la cual se realizan cese de exención de pago (cese de una exención y cese de una condonación).

Cada tipo de cese posee comportamientos distintivos (la forma de actualizar el file varía) e iguales (todos deben generar los períodos desde la fecha de vigencia en adelante) respecto a los otros tipos.

Mediante la utilización de herencia se puede definir una superclase que englobe el comportamiento común y que deje que las subclases realicen el comportamiento específico utilizando mensajes polimórficos, quedando un esquema del siguiente formato:



En el centro de cómputo, debido a la ausencia de la característica de herencia y de polimorfismo existieron dos alternativas posibles:

- *Generar tantos programas como tipos de cese: provocamos redundancia de código.*
- *Generar un único programa que englobe el comportamiento de los diferentes tipos de cese, separando el comportamiento distintivo mediante un case: provoco la generación de programas de mayor tamaño.*

Otro ejemplo útil para comprender como afecta la ausencia de herencia en el aumento del tamaño de los programas, es el expuesto en la sección 17.6.3 de este capítulo.

17.9- Errores en la aplicación.

Los errores en el sistema son difíciles (sino imposible) de evitar, la pregunta es que tipos de errores pueden ocurrir y que aplicación tiene mayor probabilidad de error.

En las secciones subsiguientes evaluaremos los siguientes aspectos:

- *Probabilidad de errores estructurales por redundancia u omisión.*
- *Probabilidad de errores conceptuales.*
- *Tiempo en que se detecta el error.*

17.9.1-Probabilidad de error por omisión o redundancia estructurales.

Cuando hablamos de errores por omisión o redundancia estructurales, estamos hablando de los errores que ocurren cuando:

- *Omitimos un campo o atributo (dependiendo del ambiente relacional u orientado a objetos) necesario: su incorporación implica la revisión exhaustiva del sistema, pues esta falta puede ocasionar errores conceptuales y no de ejecución (los programas pueden seguir corriendo en versiones antiguas que no provocan errores de ejecución y el operador no se da cuenta de que el sistema está funcionando mal).*
- *Declaramos un campo o atributo cuyo valor se deduce de otra parte del modelo: puede producir inconsistencias de datos si modificamos el campo derivado sin modificar los campos por los cuales se deduce y viceversa (ejemplo: tenemos un campo edad y otro fecha de nacimiento y modificamos la edad sin modificar la fecha de nacimiento o viceversa).*
- *Declaramos campos superfluos o no importantes: trae aparejado el aumento del volumen de la base de dato, pero no provoca el mal funcionamiento del sistema.*
- *Declaramos un campo o atributo con una estructura incorrecta: implica la realización de un estudio muy minucioso de la aplicación para dimensionar el impacto en el sistema, pero frecuentemente un cambio de formato, ocasiona errores de ejecución que son detectados y reportados por el operador (Ver ejemplo en sección 17.7.3.1).*

Como ya mencionamos anteriormente, en el centro de cómputos, la estructura de los datos es definida por los analistas, después de generar el diagrama global de la aplicación y justo antes de realizar el análisis de cada módulo.

Los analistas, en esta etapa del desarrollo, no poseen un conocimiento cabal de la aplicación que van a realizar (solo han tenido charlas con el usuario y han imaginado en sus mentes como realizar el sistema). Esta falta de experiencia, propicia la equivocación al momento de la creación de la estructura de datos.

En el proceso unificado la creación o no de los atributos de una clase, no se formaliza sino hasta que culmina la etapa de análisis y diseño del ciclo de desarrollo. La generación de la estructura de datos es paulatina, surge a través de los diferentes diagramas de análisis y diseño que desarrollamos, y es modificada a medida que adquirimos más conocimientos de la aplicación. Por ejemplo, en las distintas versiones de los diagramas conceptuales del primer ciclo de desarrollo de esta tesis, podemos apreciar que constantemente hemos incorporado o eliminado atributos de los conceptos. Esto se debió a que a medida que adquiríamos conocimiento sobre la aplicación, nos dábamos cuenta de omisiones o redundancias. Podemos concluir que es más difícil la equivocación en este proceso, ya que la estructura del sistema recién la damos por finalizada, cuando estamos terminando el análisis y vemos que todas las "piezas encajan" (es una estructura más justificada).

17.9.2-Probabilidad de errores conceptuales.

Cuando hablamos de errores conceptuales estamos hablando de errores en el entendimiento de los requerimientos del sistema. Estos errores provocan que el sistema no realice las funciones que se requieren de él o que realice funciones que no fueron solicitadas.

Creemos que el proceso unificado de desarrollo propicia la generación de aplicaciones con pocos errores conceptuales, debido a:

- ***La división en miniproyectos o ciclos del proceso unificado de desarrollo de software:*** *con esta división, logramos abordar mejor la complejidad de una aplicación, al particionarla en módulos más pequeños y manejables. Los desarrolladores examinan unas pocas funciones a la vez (las más significativas en la etapa en que nos encontramos), dejando del lado a otras (las que en el ciclo corriente no son tan significativas). De esta manera, se concentran mejor en la forma de realizar las tareas asignadas para el ciclo en estudio, abstrayéndose de los detalles irrelevantes, y disminuyendo así la posibilidad de errores.*

  En el centro de cómputos no existe la división en ciclos realizándose todo el análisis de una sola vez. Esto causa que los analistas, abrumados por la cantidad de información que deben manejar, tengan más probabilidad de omitir funciones deseadas o de no interpretar bien los requerimiento del usuario.

- ***Comunicación con el usuario:*** *Además de la lectura de los casos de uso, los usuarios experimentados deberán probar las versiones beta que resultan de cada ciclo del proceso unificado. Esto provoca que los usuarios, sepan lo que estamos realizando y den su consentimiento al funcionamiento del sistema, no teniendo la sorpresa de que luego de implementada toda la aplicación nos encontremos con que no realiza lo que se espera de ella. Mediante esta metodología lo más que podemos perder es el tiempo de realización de un ciclo.*

  En el centro de cómputos, los desarrolladores muestran el sistema al usuario cuando ya se encuentra todo implementado. Entonces corremos el riesgo que no realice las tareas que el cliente requiere de él. Si así fuese se deberá modificar el programa, provocando (muchas veces) la restructuración del mismo. (Ver ejemplo de sección 17.9.4.).

17.9.3- Tiempo en que se detectan los errores.

La naturaleza iterativa e incremental el proceso unificado de desarrollo, provoca la detección temprana de errores. Como ya expresamos el proceso unificado divide la aplicación a desarrolla en ciclos, cada uno de los cuales pasa por las diferentes etapas de desarrollo: análisis, diseño, implementación, prueba del sistema y entrega de versión al usuario. Así los desarrolladores resolverán, probarán y mostrarán las funcionalidades de un ciclo al usuario, para luego pasar al siguiente ciclo que sumará nuevas incrementos al sistema. Cuando una versión beta es entregada al usuario, este prueba el sistema determinando en este momento si el funcionamiento es correcto o no. En caso de no serlo, se procederá a repensar el ciclo. De esta manera el error será subsanado antes de concluir con la codificación de toda la aplicación.

Por el otro lado, los diferentes modelos realizados durante un ciclo del proceso unificado, provoca que la estructura del sistema sea muy pensada. A través del desarrollo de cada modelo, los analistas adquieren conocimiento de la aplicación, teniendo más posibilidades de darse cuenta de errores antes de codificar el programa.

Por ejemplo, en esta tesis durante la realización de los diagramas de colaboración del capítulo 7, nos dimos cuenta que la operación de sistema "ImputarPagos" poseía un error conceptual debido a que englobaba 2 funcionalidades bien definidas. Luego en el capítulo 8 subsanamos este error revisando todos los modelos del primer ciclo desde el caso de uso hasta el diagrama de colaboración. El error conceptual, fue detectado antes de que alguna línea de código fuese escrita.

17.9.4- Ejemplo de detección de errores: Programa de anulación de movimientos

El programa de anulación de movimientos fue realizado recientemente en la aplicación. Su creación requirió de 2 módulos de 1000 líneas cada uno en su nacimiento. A medida que lo iba codificando, el programador (con conocimiento de la aplicación) se daba cuenta de nuevos casos que en el análisis no contemplaba. Después de consultar con los analistas, el programador codificó estos casos omitidos sin quedar los mismos plasmados en el análisis. Además, cuando ya se estaba por concluir el trabajo, el programa sufrió una modificación que llevó a su reestructuración. El programa resultante no solo quedó en 2 módulos no cohesivos (por su tamaño y la tremenda cantidad de funciones distintas que realiza), sino que se hizo imposible realizar un lote de prueba que contemplara todos los casos posibles. Así que fue entregado al usuario, el cual, con el uso encuentro varios errores que se arreglaron, a medida que fueron reportados; hasta que finalmente, con el tiempo, se obtuvo un módulo óptimo.

Si se hubiese analizado mediante la metodología del proceso unificado:

1- Debido a la complejidad de este módulo, se habría dividido en 2 o más ciclos de desarrollo, para lograr abordar mejor el problema.

2- Al dividirlo en ciclos, el cambio conceptual que ocasionó la reestructuración del programa cuando ya estaba casi terminado, hubiese sido detectado al final del primer ciclo cuando mostráramos el producto al usuario.

3- Muchos errores (por falta de prueba) pudieron ser evitados ya que con la división en miniproyecto, tendremos funciones más chicas y en consecuencia más manejables y fáciles de probar.

17.10- Prueba de la aplicación

En el centro de cómputos la prueba del sistema se realiza de la siguiente manera:

1- *El programador va probando cada nueva funcionalidad que incorpora al programa que está codificando.*
   Por ejemplo:
   - *el programador genera las pantallas y codifica la forma de bifurcar*
   - *prueba el programa*
   - *incorpora luego al programa la validación de los campos*

- *prueba el programa*
- *y así sucesivamente.*

2- *Una vez concluida la codificación de un programa en su totalidad, el programador prueba el módulo para verificar que realice los requerimientos especificados por el análisis. Cumplimentada esta tarea, entrega el programa a los analistas para que ellos verifiquen.*
3- *Los analistas realizan lotes de prueba ayudados por sus análisis.*
4- *Se realizan la prueba de la aplicación en base a dichos lotes.*

Desventajas de la metodología de prueba del centro de cómputos:

- ***No existe la prueba individual de los módulos contenidos en un programa:*** *. el fuerte acoplamiento interno provoca que los diferentes módulos que lo componen no sean independientes entre sí, haciendo imposible la prueba individual de cada uno de ellos. El único que puede realizar pruebas "relativamente" individuales de las subfunciones de un programa es el programador y solo en el momento de la codificación (como se mostró en el punto 1 del párrafo anterior).*
La no realización de la prueba individual de los módulos provocará los siguientes inconvenientes:
  - *La detección de un error implicará que, una vez solucionado, se deba probar todo el programa para ver que la modificación realizada no afecte a las demás funciones en desarrollo.*
  - *Cuando se detecta un error es más difícil ubicar el lugar exacto donde ocurre (estamos probando muchos submódulos a la vez).*
- ***Los lotes de prueba son grandes:*** *deben abarcar todo el funcionamiento del programa (no existe la prueba individual e integral, sino que se realiza una única que engloba todo) y además los módulos que se prueban tienden a ser grandes con diversas funcionalidades.*
- ***Posibilidad de omisión de casos de casos de prueba:*** *debido a su tamaño es más probable la omisión de algún caso.*
- ***No ayudan a encontrar errores conceptuales:*** *Los analistas utilizan los análisis para desarrollar los lotes de prueba; los mismos poseen su visión sobre la aplicación y no la del usuario.*

Característica de la prueba de la aplicación desarrollada en esta tesis

- ***Se pueden dividir la prueba de la aplicación en dos:*** *probando el funcionamiento de los objetos en forma individual y luego su funcionamiento cuando se asocian con otros. Esta división constituye una ventaja con respecto a la otra metodología ya que:*
  - *Se disminuye el tamaño de los lotes de prueba y por ende, son menos engorrosos y poseemos menos posibilidad de omisión de casos.*
  - *Al detectar un error en la prueba individual del objeto es más fácil de ubicar el lugar exacto donde ocurre.*
  - *La detección de un error no implicará la prueba total del programa que se está realizando sino del objeto que estamos probando. Por ejemplo, si quisiéramos probar el programa de alta de un vehículo, lo primero que haremos es probar el funcionamiento de cada una de*

las clases asociadas en forma individual en lo que respecta a los métodos involucrados. En caso de ocurrir un error será solucionado y se probará nuevamente el método erróneo, no todo el programa de alta.

- ***Las pruebas están basadas en los casos de uso:*** *los desarrolladores leen los casos de uso para realizar los lotes de prueba. Los mismos fueron aprobados por el usuario, luego se aumenta la probabilidad de encontrar errores de requerimientos (si los hubiesen).*

17.11- Cohesión.

Realizando un estudio de los diferente módulos de la aplicación del centro de cómputos, notamos que lo que cambia entre uno y otro, es la complejidad de la tarea; pero todos tienen la particularidad de realizar funciones diversas en diferentes áreas (muestran pantallas y realizan el control de bifurcación de las mismas según la tecla presionada, acceden a los files necesarios, etc.), lo cual los hace no cohesivos.

La división de la aplicación en términos del tipo: alta o modificación o baja de un Vehículo, es una de las causas de la baja cohesión, ya que esta partición ocasiona que tareas pertenecientes a diferentes áreas sean codificadas en un único módulo. Además, los módulos no son pensados teniendo en cuenta la cohesión y el acoplamiento, sino que los analistas examinan las tareas necesarias para realizar la función requerida (por ejemplo el de alta de un vehículo) y las plasma en un escrito para que el programador las codifique.

Por el otro lado, en la aplicación desarrollada en esta tesis, hemos dividido la aplicación en objetos que interactúan entre sí para cumplimentar una tarea. Los objetos son poseedores de un comportamiento propio, y se asocian a otros objetos para lograr un comportamiento mayor. La forma de realizar esta asociación, es solicitando a otro objeto que se modifique o solicitando información de sus atributos. Un objeto no altera a otro objeto; si no que es el objeto mismo que modifica su estado. Al decir esto, lo que queremos expresar es que a un objeto se le dará la responsabilidad de realizar las tareas asociadas a los datos que él posee y no otras (ejemplo al Catálogo de Liquidación le daremos la responsabilidad de guardar u obtener liquidaciones, pues esos son los datos que él mantiene, sería inusual solicitarle a este catálogo los datos de un Vehículo). De esta manera, los mensajes de un objeto están funcionalmente asociados entre sí, en el sentido que todos, en su conjunto, muestran o modifican el estado del objeto (disminuyendo así la probabilidad de poseer dentro de una clase métodos de diversas áreas funcionales).

Por otro lado, mediante la utilización de patrones (en esta tesis utilizamos principalmente los patrones GRASP[Larman97]) logramos una asignación de responsabilidades más justificada y pensada, evitando en la medida posible la baja cohesión y el alto acoplamiento de las clases.

Por último, a medida que adquirimos más conocimiento de la aplicación y debido a la característica iterativa del proceso unificado, mejoramos los diferentes modelos de manera de (entre otras cosas) aumentar la cohesión de las clases. Así, por ejemplo, en la sección 15.5.1 vemos que el diagrama de colaboración de ImputarSiguiente fue modificado para aumentar la cohesión de Oficina. Como se ve en la figura 15.9, Oficina esta asociado a PagoBatch, en cambio en la figura 15.10 podemos ver que Oficina solo se comunica con

los catálogos de la aplicación (que ella mantiene) y que el catálogo de PagoBatch se comunica con los PagosBatch. Así logramos corregir el modelo del primer ciclo, de manera de aumentar la cohesión de una clase.

Para terminar podemos concluir que ***la división de la aplicación en objetos que encapsulan su comportamiento particular, la utilización de patrones a la hora de asignar responsabilidades y el mejoramiento de los modelos a medida que adquirimos experiencia****, ayuda notoriamente a aumentar la cohesión en la aplicación desarrollada en esta tesis.*

17.12- Acoplamiento.

17.12.1- Acoplamiento en la aplicación del centro de cómputos.

Si miramos el diagrama de jerarquía del anexo "Jerarquía del sistema Online", podremos observar que ***no existe acoplamiento entre diferentes módulos de la aplicación en los primeros niveles jerárquicos****. Sin embargo,* ***en los niveles inferiores de la jerarquía el acoplamiento es fuerte*** *(generalmente no mostrados en el diagrama del anexo). Lo que queremos expresar es que, por ejemplo, los módulos de alta o de rectificación o de baja de un vehículo son independientes entre sí, la modificación de uno no altera el funcionamiento de los otros. Sin embargo cada módulo está compuesto por otros submódulos (mapas, subprogramas, subrutinas externas, etc) que lo ayudan a cumplimentar la tarea que se espera de él. Dichos submódulos están fuertemente acoplados. Por ejemplo, si modificáramos un subprograma que es llamado por el módulo de alta de un vehículo, incorporándole un nuevo parámetro, luego se deberá modificar el alta para que envíe ese nuevo parámetro al subprograma cuando lo llame.*

Este fuerte acoplamiento se puede ver claramente cuando en secciones anteriores de este capítulo explicamos que no existía la prueba individual de los módulos de un programa, sino que los analistas realizaba la prueba integral. La imposibilidad de realizar pruebas individuales, demuestra el alto acoplamiento o dependencia que existe entre ellos, con todos los problemas que ello implica (la modificación de uno implica una alta posibilidad de modificar el otro, imposibilidad de transportar módulos individuales que funcionen, etc.).

17.12.2- Acoplamiento en la aplicación desarrollada en esta tesis.

Un acoplamiento que existe y que no puede ser evitado en el paradigma orientado a objetos, es el acoplamiento que produce la herencia. En efecto, una subclase está fuertemente ligada a su superclase.

Por otra parte, el hecho de que cada clase posea un comportamiento individidual respecto a otra clase no provoca el acoplamiento, pero sí su asociación con otra clase para lograr un comportamiento mayor. Esto es inevitable, dado que una clase que no posee asociaciones con otras, realizará todas las funciones ella misma y por ende, tenderá a ser de gran tamaño y a poseer baja cohesión. Lo deseable, entonces, es lograr bajo acoplamiento, no anularlo. Luego, nuestra meta, es evitar el aumento desmesurado de asociaciones. Si una clase tiene demasiadas asociaciones tendremos una clase con alto acoplamiento (depende de muchas clases).

¿Cómo logramos bajar el acoplamiento?

- ***Mediante la utilización de patrones a la hora de asignar responsabilidades a las clases:*** *los diferentes patrones nos proveen muchas veces de diferentes alternativas a seguir que nosotros evaluamos en términos de evitar acoplamiento y aumentar cohesión. Por ejemplo, cuando debíamos evaluar la clase responsable de realizar la tarea de enviar el mensaje de crear un pago online, existían 2 clases candidatas: Liquidación y Terminal. Debido a que Liquidación tenía visibilidad sobre el PagoOnline y la Terminal para disminuir el acoplamiento de esta última clase (que no veía al PagoOnline), decidimos que la adecuada de tener tal responsabilidad era Liquidación y no Terminal (ver sección 15.7).*

  - La corrección de los modelos de análisis y diseño: *la naturaleza iterativa e incremental del proceso unificado provoca que a medida que adquirimos nuevos conocimientos, podamos corregir los modelos (antes de la codificación) de manera de mejorarlos (por ejemplo en términos de acoplamiento). El ejemplo de la modificación incorporada al diagrama de colaboración de ImputarSiguiente de la sección 15.5.1 mencionado cuando hablamos de cohesión, también es valido para este caso. Cuando le quitamos a Oficina la asociación a PagoBatch no solo aumentamos su cohesión sino que disminuimos su acoplamiento, puesto que ya no posee dependencia a la clase PagoBatch.*

Para concluir, ***podemos decir que si bien la característica de herencia que existe en los desarrollos realizados mediante el paradigma orientado a objetos, aumenta el acoplamiento entre las clases; dicho incremento es compensado con la buena asignación de responsabilidades de los objetos (con la ayuda de patrones) y con la corrección de los modelos (posibilitada mediante la característica iterativa del proceso unificado) a medida que adquirimos experiencia en la aplicación.***

17.13- Reusabilidad.

17.13.1 - Reusabilidad de la aplicación en diferentes plataformas de desarrollo.

Por la forma de realización de las aplicaciones en el centro de cómputos, no existe separación entre la plataforma de trabajo y la aplicación. Las aplicaciones se realizan pensando desde el principio y siempre durante el desarrollo, en la plataforma en que se realizarán:

- *Los análisis realizados por los analistas, contienen conceptos asociados a la plataforma de trabajo, nombrando, por ejemplo, los campos de los files, mapas, nombres de files, etc.*
- *Los análisis tienen en cuenta la interfase con el usuario (cosa que no es tratada en el proceso unificado sino hasta justo antes de la etapa de codificación y teniendo especial cuidado de que las clases del dominio no tengan conexión directa a las clases de la interfase, ver sección 9.3 de esta tesis), mostrando las pantallas y considerando la interacción con el operador. No existe separación entre la interfase y el dominio del problema.*

Esta falta de separación entre la plataforma de trabajo y la aplicación, provoca que no sea reusable. El cambio de interfase de usuario, ocasionará la reingeniería de la

aplicación, debido a que la interfase se encuentra embebida dentro de los programas de la aplicación, se encuentra en el dominio del problema.

En el proceso unificado se crea el sistema sin pensar en una herramienta o plataforma específica, teniendo como resultado una aplicación más reusable en diferentes ambientes de programación (nos abstraemos de la herramienta de programación). Los diagramas del análisis y diseño, están asociados al análisis del dominio del problema (no tienen en cuenta la interfase de la aplicación sino hasta justo antes de la codificación y en este momento utilizamos indirección para lograr la comunicación entre los objetos del dominio y los de la interfase, ver sección 9.3).

17.13.2- Reusabilidad de los módulos de una aplicación

Como ya expresamos cuando hablamos de cohesión, no puede existir reusabilidad en los primeros niveles de la jerarquía (mostrados en la jerarquía del anexo "Jerarquía del sistema"). En efecto el módulo de alta de un vehículo, por ejemplo, no puede ser reutilizado en su totalidad en otro lado de la aplicación, pero sí podría ser usado parte de su funcionamiento interno (por ejemplo el cálculo de los importes de los períodos de un vehículo). De esta manera, ***la reutilización de módulos es factible en los niveles inferiores de la jerarquía pero no en los superiores.***

La reusabilidad de los niveles inferiores puede ser detectada por:

- Programadores: *cuando no afecta globalmente a la aplicación, los programadores generan los módulos de los niveles inferiores de la "Jerarquía del sistema", dado que son considerados una parte interna de un programa. El programador divide el programa tratando de separarlo de manera de no aumentar su tamaño en demasía. En caso de considerar que una función de su programa puede ser utilizada en otro punto de la aplicación la separara de las demás. Lo que realmente se tiene en mente es el evitar la redundancia en la codificación y propicia la reutilización local o sea para la generación o modificación que se está realizando en el momento. El programador no puede realizar otra cosa, dado que no posee la visión global de la aplicación.*

Por ejemplo, debido a la incorporación de un nuevo formulario se hace necesario la incorporación de un nuevo file a la aplicación para guardar los datos del mismo. Luego diferentes programas tales como el de alta, baja o modificación de Vehículo requieren acceder a cierto dato del formulario para realizar una validación. El programador (que debe realizar la modificación de todos los programas) realiza entonces un único módulo que accede al nuevo file, el mismo es llamado por todos los programas que requieran el dato en cuestión. De esta manera, se logró la reutilización de un módulo. Nótese que, en el caso de que el todas las modificaciones de los programas que acceden al nuevo file no le hubiesen sido asignadas al mismo programador, seguramente no hubiese realizado ese módulo reusable, por no considerar que el acceso al file aumente en demasía el tamaño del programa como para justificar la generación de un subprograma y por no ver el posible uso en otros puntos de la aplicación.

- Analistas: *el análisis de un módulo del nivel inferior de la "Jerarquía de Módulos" es realizado cuando el mismo es utilizado globalmente en la aplicación. Por ejemplo, el módulo que calcula los importes de los diferentes períodos del*

impuesto, es usado por toda el sistema y por ende, dicha rutina tan crítica es razón de estudio independiente por parte de los analistas.

En la ***aplicación desarrollada en esta tesis****, diversas características ayudan a la generación de módulos reusables:*

- ***La generación de módulos más cohesivos y con menos acoplamiento:*** *Los módulos con baja cohesión realizan tareas diversas, de esta manera son más específicos y por ende poseen menos probabilidad de reusabilidad. Los módulos con alto acoplamiento están fuertemente ligados a otros, por lo cual su resusabilidad es más difícil ya que implica el reuso del módulo y de sus dependencias.*
- ***La utilización de la metodología del proceso unificado:*** *a través de los diferentes modelos y ciclos del proceso unificado podemos dilucidar comportamientos similares antes de la codificación. Por ejemplo, durante la realización del diagrama de colaboración nos dimos cuenta que el cálculo del importe de la barra y el de la liquidación constituía una única función a la cual se le pasaba como parámetro la fecha. En el centro de cómputos esta función reusable, hubiese sido descubierta por el programador en el momento de la codificación, si es que los dos importes se calcularan en un único programa, codificado por una misma persona.*
- ***Rápida ubicación de las funciones:*** *la propiedad de encapsulamiento de los datos, provoca que el comportamiento de los objetos y sus atributos sean encerrados dentro del objeto mismo.*
- ***Herencia:*** *mediante el concepto de herencia logramos que las funciones generales sean realizadas por la superclase y las más específicas por las subclases, de esta manera cada subclase no codificará el comportamiento general sino que le será suministrado por la superclase. Así el comportamiento de la superclase es reusado en cada una de las subclases.*
- ***Polimorfismo:*** *mediante la creación de operaciones polimórficas logramos que se dispare una operación dependiendo del tipo de datos que envía el mensaje. Este concepto permite sacar o poner opciones en programas sin que ello afecte al funcionamiento de la aplicación y además a través del polimorfismo podemos encapsular las funciones específicas de las subclases, logrando, de esta manera, una superclase más general (y reusable).*

17.14- Ejemplificación de las consecuencias de la ausencia del polimorfismo y herencia.

Para concluir, 2 características que hemos mencionado y que son inexistentes en el paradigma procedural son la herencia y la posibilidad de realizar operaciones polimórficas. En esta sección ejemplificaremos la utilización de estas 2 características y las consecuencias que puede causar el no poseerlas.

17.14.1- Ejemplo: Incorporación de una nueva opción al menú de Liquidación

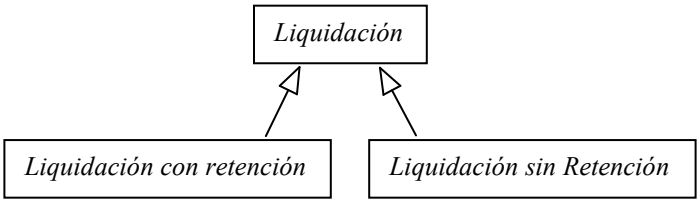
En un momento en la aplicación, se necesitó la incorporación de una nueva liquidación que tiene otras fechas de vencimiento y que refleja las retenciones que realiza el registro sobre un vehículo (para ello se muestra una pantalla con la deuda de cada período y el operador indica los períodos e importes retenidos por el registro).

Esto requiere:

- *la incorporación de una nueva opción al menú de liquidación*
- *la creación de un programa que realice la nueva liquidación*
- *la incorporación a los case del listador de la nueva opción*

Note que si quisiéramos eliminar esta opción deberíamos nuevamente modificar el programa listador (de 2000 líneas) para sacar las líneas asociadas a esta opción (ya que este módulo es el encargado de listar todas opciones de liquidación de la aplicación).

Con el paradigma orientado a objetos, seguramente tendremos un esquema de la siguiente forma:



Tanto la liquidación con retención o la sin retención constituyen una subclases de la clase liquidación y por ende, heredan los atributos y métodos de la clase liquidación, sumando además sus propias funcionalidades. Así los métodos tales como obtener el total a abonar, obtener los datos del vehículo a liquidar, serán realizados por los métodos que se encuentran en la superclase Liquidación. Por otro lado, los métodos tales como obtener la fecha de vencimiento de la liquidación serán redefinidos en cada una de las subclases (mediante la utilización de métodos polimórficos).

El código directamente asociado a la nueva opción de liquidación queda dentro de la clase liquidación con retención y el asociado a todas las opciones de liquidación queda en la Liquidación. Así facilitamos la eliminación o incorporación de nuevas opciones en el programa de Liquidación.

17.14.2- Ejemplo: Separación de programa de alta en opción

Originariamente muchos de los programas actuales constituían un único módulo que realizaba toda la tarea. Sin embargo debido a las limitaciones existentes en los parámetros de catalogación y al aumento de tamaño de los programas (provocado por continuos cambios), se llegó al punto de que una nueva modificación no pudo ser catalogada. Luego existían 2 opciones a seguir:

- *Reingeniería del programa.*
- *Dividir el módulo que no catalogaba de manera de disminuir su tamaño.*

¿Por qué no se realizó la reingeniería?

Las nuevas funcionalidades que debieron ser incorporadas tenían urgencia y la reingeniería hubiese implicado un tiempo de desarrollo y prueba que no existía.

Los módulos a rehacer estaban en funcionamiento y eran críticos y no podían ser inhabilitados a la espera del nuevo programa.

Los análisis estaban desactualizados, así que la reingeniería hubiese implicado realizar un trabajo de reingeniería inversa para no omitir funciones que no estaban debidamente documentados (lo que implicaba aún más tiempo de desarrollo).

Lo más eficiente, ante la imposibilidad de realizar una reingeniería, hubiese sido la división de los programas en módulos según sus funcionalidades. Por ejemplo el alta pudo tener un módulo que manejara las distintas pantallas y otro que manejara el acceso a la base de datos. Sin embargo, esta división requería de la búsqueda dentro del programa de las funcionalidades que irían a uno u otro módulo. Luego hubiese sido necesario una reestructuración del programa y no se disponía de tiempo, ni para reestructuración, ni para la prueba.

¿Qué se hizo?

Se dividió el programa por opciones. Por ejemplo existen varias opciones de alta:

Alta por primera inscripción, Alta por cambio de radicación, Alta por primera inscripción con retención. Estos programas tenían leves diferencias en cuanto a la forma de calculo de los períodos que originariamente estaban tratados en el mismo programa mediante una sentencia case. Se dividió el programa en 3, quedando prácticamente iguales, salvo porque cada uno tenía las consideraciones de cálculo de su opción. Con esto se logró eliminar la cantidad de líneas necesarias para que el programa ejecutara y así poder no dejar sin servicio al usuario. Esta salida se tomo con la idea de que con el tiempo, se realizaría la reingeniería del programa, pero nuevas urgencias hicieron que los programas quedarán así, y el resultado fue que:

- *La modificación de uno implica la modificación de los otros dos.*
- *Si un programa tiene error los otros dos es prácticamente seguro que lo tendrán*
- *El mantenimiento de los programas es, en el caso del alta, por triplicado*
- *Propicia a la larga a la reingeniería inversa*

Un aspecto importante a considerar es ¿cual es la verdadera causa de la división por opciones?

El impacto de la división en funcionalidades más pequeñas resulta muy grande, ya que implicaría la reprogramación y probado en su totalidad de la opción, debido al alto acoplamiento y la baja cohesión de los módulos. Esto propicia a que, cuando no hay tiempo, la división se realice por opciones. Esta alternativa es un "arma de doble filo", ya que si bien solucionamos el problema de catalogación rápidamente, disminuimos notoriamente su ciclo de vida del programa acelerando el tiempo para la reingeniería del mismo.

En el proceso unificado esto no hubiese ocurrido, ya que la forma de división de la aplicación por objetos y no por funciones provoca un encapsulamiento de los datos en los objetos disminuyendo el tamaño de la codificación. Por el otro lado, mediante la utilización de herencia y operaciones polimórficas, el módulo de alta hubiese sido pensado desde su nacimiento como una clase que realiza el comportamiento general (superclase) y varias subclases (de acuerdo a las opciones) que realizan el comportamiento específico.

De esta manera, la división del programa hubiese sido desde su generación, no cuando nos encontramos que tenemos la urgencia de disminuir su tamaño.

17.15- Conclusión final.

En esta tesis, desarrollamos el análisis y diseño de los primeros ciclos de una aplicación ejemplo "Subsistema de Liquidación y Pagos del Impuesto a los Automotores" utilizando el paradigma orientado a objetos y la metodología del proceso unificado de desarrollo.

Dicha aplicación ya se encuentra en funcionamiento y su propósito es la emisión de boletas de liquidación del impuesto y el control del pago del mismo. Sus desarrolladores utilizaron el paradigma algorítmico y la metodología de desarrollo de "Cascada".

Luego utilizando la experiencia obtenida de la realización de la aplicación en ambas metodologías de desarrollo, las comparamos.

Para lograr dilucidar las ventajas de la utilización del proceso unificado y del paradigma orientado a objetos, estudiamos ambos sistemas en términos de: facilidad de adquisición de conocimiento, redundancia de información en ambas aplicaciones, estimación, impacto y factibilidad de modificación del sistema, tamaños de los módulos de ambas aplicaciones, errores en ambas aplicaciones, cohesión, acoplamiento, reusabilidad y herencia.

A través de este estudio, logramos dilucidar que la generación de la aplicación mediante la utilización de la metodología del proceso unificado y el paradigma orientada o objetos, provoca la creación de módulos con las siguientes características:

- *Poseen alta cohesión.*
- *Poseen baja acoplamiento.*
- *Son más fáciles de modificar.*
- *Son más fáciles de probar.*
- *Es más probable su reusabilidad ya sea en diferentes plataformas como en la misma aplicación.*
- *Poseen menor tamaño.*
- *El código es más fácil de ubicar debido a que se encuentra encapsulado en el objeto asociado.*

De esta manera, a través del estudio y desarrollo de ambas aplicaciones, logramos vislumbrar las ventajas que acarrea la utilización del proceso unificado de desarrollo y del paradigma orientado a objetos para realizar un sistema.

Anexo "Casos de uso"

Para obtener un diagrama que caso de uso existen varios pasos a seguir:

1- *Definir los límites del sistema*
2- *Identificar los actores*
3- *Identificar los casos de uso*
4- *Escribir todos los casos de uso en formato de alto nivel*
5- *Categorizar los casos de uso como primarios, secundarios u opcionales*
6- *Dibujar un diagrama de caso de uso*
7- *Relacionar los casos de uso e ilustrar relaciones de casos de uso*
8- *Escribir los casos de uso de mayor riesgo o influencia o críticos, en forma esencial expandida. Esto se hace para un mejor entendimiento del problema y para estimar la naturaleza y longitud del sistema*
9- *Tratar de delegar los casos de uso reales hasta la fase de diseño del ciclo de desarrollo, dado que su creación involucra decisiones de diseño.*

   Causas por las cuales justifica una definición de casos de usos reales

   9.1) Su descripción ayuda significativamente a la comprensión del problema.

   9.2) Los clientes demandan especificar sus procesos de esta manera.

1- Definir los límites del sistema

¿Porqué definir el límite de un sistema?

1- Para capturar los requisitos correctos y construir el sistema que satisface las necesidades reales del usuario, los desarrolladores requieren de un firme conocimiento del contexto del sistema.

2- Una vez elegido el límite del sistema, el ambiente externo está representado solamente por los actores.

2- Identificación de los actores

¿Qué es un actor?

Es una entidad externa (no necesariamente un ser humano) quien en algún camino participa en el desarrollo de un caso de uso.

Un actor estimula al sistema con eventos de entrada y recibe algo de él. Representan terceros fuera de la aplicación que estamos desarrollando que colaboran con ella.

Los actores utilizan el sistema interactuando con los casos de uso, y son identificados por los roles que ellos cumplen en los casos de uso.

Para todo caso de uso, hay un actor iniciador que genera el estímulo de comienzo y posiblemente hay otros actores que participan en el caso de uso.

La identificación de los casos de uso y los actores es una tarea clave para obtener los requisitos correctos (aquellos que necesita el usuario)

Clases de actores

1- *Roles que cumplen personas (contribuyente del Impuesto, operador del sistema)*
2- *Sistema de computadoras (sistema de autorización de tarjeta de crédito, sistema de autorización de cuenta bancaria).*
3- *Dispositivo mecánico o eléctrico (un timer en un sistema de control de riego).*

Criterios para la elección de Actores

1- *Debería ser posible identificar un usuario como actor candidato. Esto nos ayudará a encontrar los actores relevantes y eliminará los actores innecesarios (aquellos que nosotros solo imaginamos que existen).*
2- *Debería haber cierta coherencia entre los actores y los roles que representan. Lo que se quiere decir con esto, es que no debería existir solapamiento de actividades entre actores, o sea actores que realicen las mismas tareas.*

3- Identificación de los casos de uso

¿Qué es un caso de uso?

Es un documento narrativo que describe las secuencias de eventos de un actor (un agente externo) usando el sistema para completar un proceso.

Es una secuencia de acciones que el sistema lleva a cabo para ofrecer algún resultado de valor a un actor.

A través de los casos de uso capturamos los requisitos funcionales del sistema y los requisitos no funcionales (que son especificados para cada caso de uso). Cada usuario quiere que el sistema realice algo para él o ella, es decir lleve a cabo ciertos casos de uso. Para él o ella un caso de uso es una manera de utilizar el sistema. En consecuencia, si se logra describir todos los casos de uso que necesita el usuario, entonces sabremos que es lo que debe hacer el sistema. Cada caso de uso representa una forma de utilizar el sistema. Un usuario puede necesitar varios casos de uso distintos para realizar sus tareas, por lo tanto cada usuario puede representar a uno o varios actores dependiendo de los distintos casos de uso.

¿Porqué los casos de uso?

Los casos de uso son las funciones que proporcionan un sistema para añadir valor a sus usuarios. Tomando la perspectiva de cada usuario podemos capturar los casos de uso que necesita para realizar el trabajo.

La estrategia del caso de uso es reformular la pregunta típica para saber las funciones del sistema añadiéndole tres palabras:

*¿Qué debe realizar el sistema **para cada usuario**?*

Estas tres palabras nos hacen mantener centrados en las necesidades del usuario, nos hacen pensar en las funciones que cada usuario necesita. También nos ayuda a abstraernos de funciones que solamente imaginamos nosotros que pueden ser necesarias, funciones superfluas que el usuario en realidad no usa.

Ventajas de los casos de uso

- *Los casos de uso ayudan a los desarrolladores, usuarios y clientes a llegar a un acuerdo sobre la forma de utilizar el sistema.*
- *Los casos de uso son intuitivos, o sea que los usuarios y los clientes no tienen que aprender una notación compleja.*
- *Proporcionan una manera fácil de organizar los requisitos del sistema. Todos los requisitos funcionales quedan asociados a los casos de uso y también muchos de los requisitos no funcionales.*
- *Los desarrolladores pueden dividir el trabajo de captura de requisitos entre ellos y después utilizar los resultados como entrada al análisis, diseño, implementación y prueba.*

Criterios útiles para la identificación de un caso de uso

Hay dos conceptos claves que constituyen criterios útiles para la identificación de los casos de uso:

- *Resultado de valor: Cada ejecución satisfactoria de un caso de uso debe proporcionar algún valor para alcanzar un objetivo. Este concepto nos ayuda a evitar casos de uso demasiado pequeños.*
- *Un actor concreto: identificar casos de uso que den valor a usuarios individuales reales. Este concepto nos asegura que no serán demasiado grandes.*

Método para identificar un caso de uso

Basados en actores:

- *Identificar los actores relacionados al sistema u organización*
- *Para cada actor identificar los procesos que ellos inicializan*

Basados en eventos:

- *Identificar los eventos externos que el sistema debe responder*
- *Relacionar los eventos a actores y casos de uso*

4- Descripción de todos los casos de uso en un formato de alto nivel

Tipo de Casos de Uso

- ***Esenciales:*** *son aquellos que se representan en forma ideal, permaneciendo su descripción libre de tecnología y de detalles de implementación. Las decisiones de diseño se difieren y se abstraen (especialmente aquellas relacionadas a la interfase). Los casos de uso esenciales describen los procesos en término de sus actividades y motivaciones esenciales.*
- ***Reales:*** *son utilizados para especificar tecnología de entrada/salida, ya que describen los procesos en término del diseño corriente real.*

Formatos de los Casos de Uso

- ***Formato de alto nivel****: Se describe el proceso muy brevemente, en 2 o 3 sentencias. Es útil para entender en forma fácil el grado de complejidad y funcionalidad del sistema*

- ***Formato extendido:*** *Se describe el proceso en forma mucho más extendido. Posee una sección de eventos en donde se los detalla paso a paso y se describe la forma de respuesta que debería tener el sistema cuando ellos ocurren.*

6- Diagrama de caso de uso.

Un diagrama de caso de uso ilustra el conjunto de casos de uso de un sistema, sus actores y la relación de los actores con los casos de uso. Estos diagramas facilitan que los sistemas y subsistemas sean abordables y comprensibles al presentar la vista externa de cómo puede utilizarse en un contexto dado. Son importantes ya que permiten visualizar el comportamiento del sistema de forma que los usuarios puedan ver como utilizar esos casos de uso y de forma que los desarrolladores puedan implementarlos.

El diagrama de caso de uso cubre principalmente el comportamiento del sistema: los servicios visibles externamente que proporciona el sistema en el contexto de su contorno.

¿Para qué se utilizan los diagramas de casos de uso?

1- Para modelar el contexto de un sistema.

Al dibujar una línea alrededor de los casos de uso, separándolos de los actores, identificamos claramente cual es el límite del sistema. Además este diagrama nos permite identificar los actores y el significado de sus roles.

2- Para modelar los requisitos de un sistema.

Un requisito es una característica de diseño, una propiedad o un comportamiento del sistema. Cuando se enuncian los requisitos de un sistema se está estableciendo un contrato entre elementos externos y el propio sistema, que establece lo que se espera que haga el sistema. La mayoría de las veces no importa como lo hace, sino que lo realice. Un sistema con un comportamiento correcto llevará a cabo todos los requisitos de manera fiel, predecible y fiable. Al construir un sistema es importante, que al principio se realice un acuerdo de lo que debería hacer el sistema, aunque, con total seguridad, la comprensión de los requisitos, evolucionará conforme se vaya implementando el sistema de manera incremental e iterativa. Los requisitos funcionales de un sistema se pueden expresar con casos de uso y los diagramas de caso de uso son fundamentales para manejar esos requisitos.

Especificar los requisitos implica especificar que debería realizar el sistema (desde el punto de vista externo), independientemente de cómo se haga. Se utilizarán los diagramas de casos de uso para especificar el comportamiento deseado del sistema. De esta forma, el diagrama de caso de uso nos permite ver el sistema entero como una gran caja negra, se puede ver que hay fuera del sistema y como reacciona a los elementos externos, pero no se puede ver como funciona por dentro.

7- Relaciones entre de casos de uso.

1- ***Generalización:*** *lo que significa es que el caso de uso hijo hereda el comportamiento y significado del caso de uso padre. El hijo puede*

añadir o refinar el comportamiento del padre, el hijo puede ser colocado en cualquier lugar donde este el padre.

2- ***Inclusión:*** *esta relación entre casos de uso significa que un caso de uso base incorpora explícitamente el comportamiento de otro caso de uso. La inclusión puede verse como que el caso de uso base toma el comportamiento del caso de uso proveedor. Se usa para evitar describir el mismo flujo de eventos repetidas veces, poniendo el comportamiento común en un caso de uso aparte. Básicamente se toma un conjunto de responsabilidades del sistema y se capturan en un sitio (el caso de uso a incluir en otros) y a continuación se permite que otras partes del sistema (otros casos de uso) incluyan el caso de uso siempre que necesiten las funcionalidades que él provee.*

3- ***Extensión:*** *esta relación entre casos de uso significa que un caso de uso base incorpora implícitamente el comportamiento del otro caso de uso en el lugar especificado indirectamente por el caso de uso que extiende al base. El caso de uso base puede extenderse solo en ciertos puntos, llamados puntos de extensión. El caso de uso puede estar aislado pero bajo ciertas condiciones, su comportamiento puede extenderse con el comportamiento del otro caso de uso. La extensión puede verse como que el caso de uso que se extiende incorpora su comportamiento en el caso de uso base. Se usa para modelar casos de uso con comportamientos opcionales, de manera que se separa ese comportamiento al obligatorio. También son utilizados para modelar subflujos que se ejecutan solo bajo ciertas condiciones o para modelar varios flujos que pueden insertarse en un momento dado.*

Anexo Diagramas de Colaboración

Conceptos acerca de interacciones y diagramas de interacción.

Una interacción es un comportamiento que incluye un conjunto de mensajes intercambiados por un conjunto de objetos dentro de un contexto para lograr un propósito. Las interacciones se utilizarán para modelar los aspectos dinámicos de un sistema o sea la forma en que los objetos interactúan entre sí y toman roles de manera de lograr un comportamiento mayor al que pueden realizar en forma individual.

Cada interacción puede modelarse de dos formas:

- *destacando el ordenamiento temporal de los mensajes (diagrama de secuencia)*
- *destacando la secuencia de mensajes en el contexto de una organización estructural de objetos (diagrama de colaboración)*

Una interacción tiene una naturaleza estática ya que establece un escenario para un comportamiento del sistema introduciendo todos los objetos que colaboran para realizar alguna acción. La interacción incluye un conjunto de mensajes enviados entre objetos. Un mensaje constituye la especificación de la comunicación entre objetos que transmiten información, esperando desencadenar una actividad.

Cuando se modela una interacción lo que se está haciendo es construir una representación gráfica de las acciones que tienen lugar entre un conjunto de objetos.

Los diagramas de secuencia y los de colaboración constituyen diagramas de interacción. Ellos son prácticamente isomorfos entre sí, o sea se puede partir de uno de ellos y transformalos en el otro sin perder información.

Gráficamente, un diagrama de secuencia es una tabla que representa objetos (dispuestos a lo largo del eje X) y mensajes ordenados según se suceden en el tiempo (en el eje Y). Por el otro lado, gráficamente, un diagrama de colaboración es una colección de nodos y arcos.

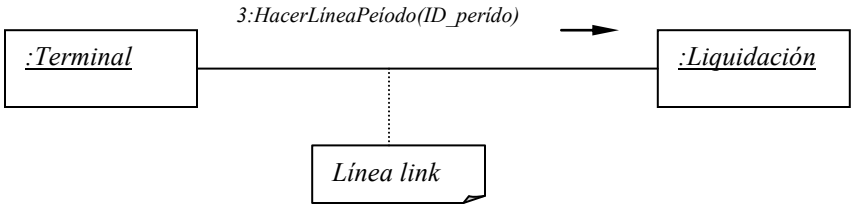
En este anexo se dará una noción de la forma de ilustrar los diagramas de colaboración.

Sintaxis de los diagramas de colaboración.

A continuación, mostraremos las distintas ilustraciones que pueden aparecer en un diagrama de colaboración:

Liquidación	*Estamos haciendo referencia a la **clase** Liquidación*
:Liquidación	*Estamos haciendo referencia a una **instancia** de la clase Liquidación*
L1:Liquidación	*Estamos haciendo una **instancia nombrada** de la clase Liquidación*

Un link es camino de conexión entre 2 instancias que indica alguna forma de navegabilidad y visibilidad. Más formalmente un link es una instancia de una asociación. Mirando a dos instancias como una relación Cliente/Servidor , un link (o camino de navegación) nos dice que pueden ser enviados mensajes del Cliente a el Servidor.



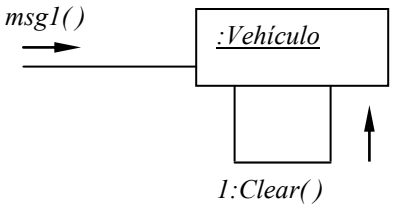
Como se ve en este ejemplo los mensajes son representados a través de un nombre junto con una flecha sobre la línea que representa el link entre los objetos.

La sintaxis de un mensaje es la siguiente:

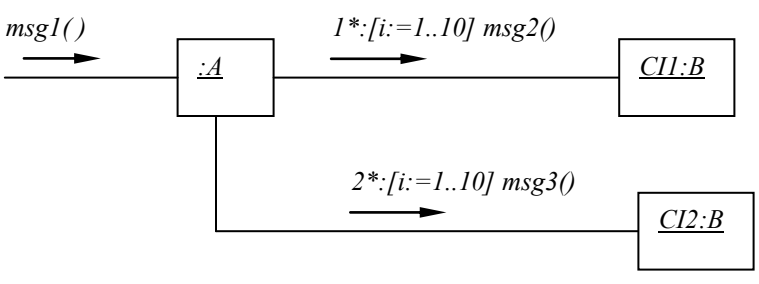
Secuencia: Valor de retorno:= nombre del mensaje(parametros: tipo) : tipo de retorno

Donde
Secuencia: indica el orden en que se va a realizar el mensaje dentro de la interacción (el primer mensaje del diagrama de colaboración no posee número.
Valor de retorno: es el nombre de la variable del valor retornado
Parametros: se pueden especificarse varios parametros separados por coma. El tipo del parámetro puede o no ser especificado
Tipo de retorno: puede o no ser especificado

*Tener en cuenta que **un mensaje puede enviar un mensaje a sí mismo**, en tal caso, se visualizará de la siguiente forma:*

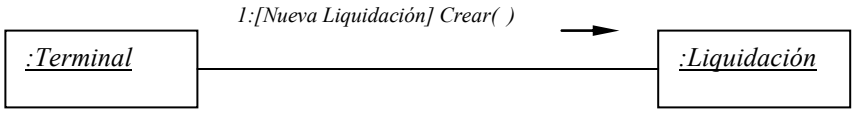


*Una **iteración** indica que el mensaje es enviado repetidamente, en un loop, al receptor. Gráficamente, es indicada poniendo un "*" a continuación del número de secuencia. Posee una opción que permite ingresar la cantidad de veces que se realizará una iteración. Si se desea expresar que más de un mensaje se repite con la misma iteración repetir la opción de cantidad de veces para cada mensaje*

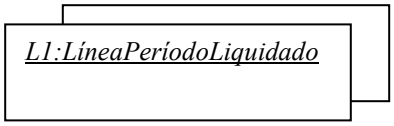



```
Msg1()
{
For i = 1 to 10
     {
     CI1.msg2( )
     CI2.msg3( )
     }
}
```

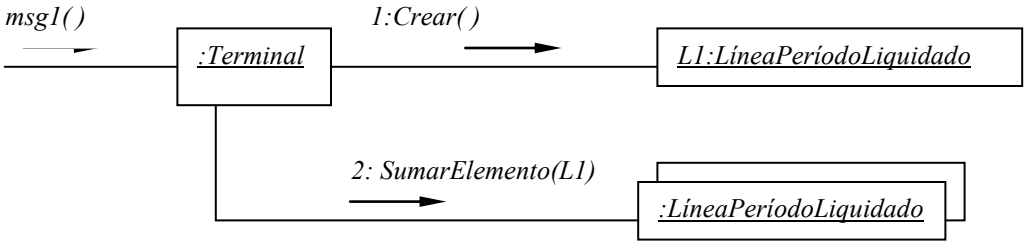
*Un **mensaje condicional** es ilustrado poniendo la condición por la cual se realizará el mensaje, después del número de secuencia entre corchetes. Si no se cumple dicha condición el mensaje no será realizado.*

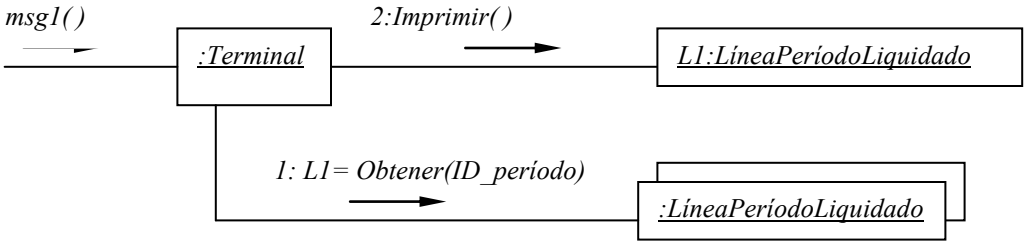


*Un **multiobjeto** representa un conjunto de instancias lógicas y es representado de la siguiente manera:*



Tener en cuenta que cuando se envía un mensaje a un multiobjeto, el mensaje es en realidad enviado a la colección





Anexo "Patrones GRASP"

Responsabilidades

Booch define las responsabilidades como "un contrato y obligación de un tipo o clase". Las responsabilidades están relacionadas a las operaciones de un objeto en término de su comportamiento.

Existen dos tipos de responsabilidades:

- *asociadas a lo que es el objeto*
- *asociadas a lo que conoce el objeto*

La responsabilidad del primer tipos ocurren cuando:

- *se hace algo a sí mismo*
- *inicializa acciones en otro objeto*
- *controla y coordina actividades en otros objetos*

Las responsabilidades de segundo tipo ocurren cuando:

- *se conoce acerca de los datos*
- *se conoce acerca de los objetos relacionados*
- *se conoce acerca de las cosas que se pueden derivar o calcular*

Las responsabilidades son asignadas a los objetos durante el diseño orientado a objetos. Mediante el diagrama conceptual podemos deducir casi todas (si no todas) las responsabilidades relacionadas al conocimiento.

Las responsabilidades no son lo mismo que los métodos:

Una responsabilidad es implementada por uno o varios métodos los cuales actúan solos o en colaboración de otros métodos u objetos. Son asignadas durante la creación del diagrama de interacción. Podemos decir que los diagramas de interacción muestran la elección de la asignación de responsabilidades a los objetos.

Patrones

Existe un repertorio de principios generales de soluciones idiomáticas que nos guían en la creación de aplicaciones de software. Dicho repertorio nos provee de una ayuda para realizar el proceso de asignación de responsabilidades a objetos que poseen la misma categoría de problemas. Si estos principios son decodificados de manera estructurada describiendo el problema y la solución al mismo, entonces pueden ser llamados patrones.

Un patrón es una descripción nombrada de un problema y su solución, que puede ser aplicada en diferentes contextos a veces con consejos de cómo aplicarlos en situaciones nuevas.

Los patrones no intentan descubrir y expresar nuevos principios de ingeniería de software, sino que desean codificar los conocimientos, modismos y principios existentes, los más usados y asentados.

Los patrones G.R.A.S.P. [Larman97] describen principios fundamentales para asignar responsabilidades a objetos, expresados como patrones.

Los patrones G.R.A.S.P. son:

1- *Experto*
2- *Creador*
3- *Alta cohesión*
4- *Bajo acoplamiento*
5- *Controlador*
6- *Polimorfismo*
7- *Fabricación Pura*
8- *Indirección*
9- *No hablar con extraños*

Para mayor información sobre estos patrones remitirse al libro "Applying UML y Patterns" publicado por Craig Larman [Larman].

1- Patrón Experto.

Problema: *¿Cuál es el principio más básico para el cual las responsabilidades son asignadas en el diseño orientado a objetos?*

Solución: *Asignar la responsabilidad al experto de la información osea a la clase que tiene la información necesaria para completar la responsabilidad.*

Generalidades

Este patrón expresa la intuición de que los objetos hacen cosas relacionadas a lo información que ellos tienen.

Muchas veces para completar una responsabilidad se requerirá de información que se extiende a través de distintas clases de objetos. Esto implica que existen muchos expertos parciales que colaboran para lograr realizar la tarea.

Experto nos guía a ver donde los objetos de software realizan las operaciones. Generalmente se hacen las cosas del mundo real que ellos representan. Osea que es el objeto mismo el que realiza sus operaciones. Existe una distinción entre los objetos de software y los del mundo real. Estos últimos son objetos inanimados, mientras que los primeros son animados (ellos pueden tomar responsabilidades y hacer cosas respecto a la información que conocen).

Beneficios

- *Bajo acoplamiento: los objetos usan su propia información para completar la tarea y esto provoca que el encapsulamiento se mantenga.*
- *Tiende a tener alta cohesión: el comportamiento es distribuido a través de las clases que tienen la información requerida, de esta manera las definiciones de clase tienden a ser más ligeras, lo que hace que sean más fácil de mantener y entender.*

2- Patrón Creador.

Problema: *¿Quién sería responsable de crear una nueva instancia de alguna clase?*
Solución: *Asignar a la clase B la responsabilidad de crear una instancia de la clase A si:*

- → *B agrega objetos de A*
- → *B contiene objetos de A*
- → *B registra instancias del objeto A*
- → *B usa objetos de A*
- → *B tiene la inicialización de los datos que serán pasados a A*
- → *B es un creador de objetos de A*

Generalidades

Creador guía en la asignación de responsabilidades relacionadas a la creación de objetos. El intento de este patrón es encontrar un creador el cual necesite ser conectado al objeto creado vía un evento. Creador sugiere que la clase que registra o contiene es buena candidata para tener la responsabilidad de crear esa cosa.

Beneficios

- *El acoplamiento no se incrementa porque la clase creada ya es visible por el creador (luego hay asociaciones ya existentes que motivaron la selección de ese experto)*

3- Patrón de Bajo Acoplamiento.

Problema: *¿Cómo soportar baja dependencia y así incrementar el reuso?*
Solución: *Asignar la responsabilidad de manera que el acoplamiento permanezca bajo.*

Generalidades

- *El acoplamiento es la medida de la fuerza de la asociación establecida por una conexión entre los módulos (entre clases). Una clase con bajo acoplamiento implica que no depende de otras clases.*

Desventajas de acoplamiento fuerte:

- *Las clases son más dificiles de comprender, cambiar o corregir si están muy interrelacionadas con otros módulos.*
- *Dificulta la reusabilidad: dado que el uso de una clase fuertemente acoplada requiere la presencia adicional de las clases de las cuales ella depende.*
- *Los cambios en clases relacionadas fuerza a cambios locales.*

Existe una tensión entre el concepto de herencia y acoplamiento, porque la herencia introduce un acoplamiento considerable. Por un lado, son deseables clases débilmente acopladas, por el otro la herencia (que acopla fuertemente las superclases y subclases) ayuda a explotar las características comunes de las abstracciones.

Este patrón no puede ser considerado independientemente de otro patrón (tales como experto o creador), pero es uno de los principios que influye en la selección de asignación de responsabilidades.

El acoplamiento muy bajo no es deseable, ya que consideremos el principio fundamental de la tecnología de objetos:

- *un sistema de objetos conectados que se comunican vía mensajes*

Si tengo muy poco acoplamiento, nos lleva a un diseño pobre porque esta conducido por objetos complejos que realizan todo el trabajo.

Beneficios

- *Los módulos no se afectan por el cambio en otros módulos.*
- *Los módulos son simples de entender aisladamente.*
- *Los módulos son más reusables.*

4- Patrón de Alta Cohesión.

Problema: *¿Cómo conservar manejable la complejidad?*
Solución: *Asignar la responsabilidad de manera que la cohesión se mantenga alta.*

Generalidades

La cohesión mide el grado de conectividad entre los elementos de un solo módulo (en diseño orientado a objetos, mide el grado de conectividad entre los elementos de una clase u objeto).

Existen distintas forma de cohesión:

- *cohesión por coincidencia: se incluyen en la misma clase o módulo abstracciones sin ninguna relación (esta es la menos deseable)*
- *cohesión funcional: los elementos de la clase o módulo trabajan todos juntos para proporcionar algún comportamiento bien delimitado*

En término de diseño orientado a objetos la cohesión (o más precisamente, la cohesión funcional) es la medida de cómo se relacionan y enfocan las responsabilidades en una clase.

Una clase con responsabilidades altamente relacionadas y que no tiene una cantidad tremenda de trabajo, tiene alto cohesión. Grady Booch describe la cohesión funcional alta cuando los elementos de un componente (tal como una clase) "todos trabajan juntos para proveer un comportamiento bien delimitado".

Una clase con baja cohesión tiene cosas no relacionadas o relacionadas con mucho trabajo. No son deseables dado que son difíciles de comprender, de reusar y mantener. Estas clases toman responsabilidades que tendrán que delegar a otras.

Grados de cohesión:

- *Muy Bajo: una clase tiene la responsabilidad de muchas cosas en diferentes áreas funcionales.*
- *Baja: una clase tiene responsabilidad de tareas complejas.*
- *Moderada: una clase posee pocas responsabilidades en pocas áreas que son lógicamente relacionadas a un concepto de clase, pero no una a la otra.*

- *Alta: una clase tiene responsabilidad moderada y colabora con otras clases para satisfacer la tarea.*

Beneficios de alta cohesión

- *La claridad y facilidad de compresión del diseño es incrementada*
- *El mantenimiento y mejoramiento son simplificados*
- *Soporta bajo acoplamiento*
- *Reusabilidad*

5- Patrón Controlador.

Problema: *¿Quién sería responsable de manejar los eventos del sistema?*

Solución: *Asignar la responsabilidad para manejar un evento de sistema a la clase que represente alguna de las siguientes condiciones:*

- *Representa todo el sistema (controlador Facade).*
- *Representa todo el comercio u organización (controlador Facade).*
- *Representa alguna cosa del mundo real que es activa (por ejemplo el rol de una persona) que puede estar involucrada en la tarea (Controlador de roles).*
- *Representa un manejador artificial de todos los eventos de un caso de uso, usualmente llamado "<Nombre del caso de uso>Manejador" (Controlador de caso de uso).*

Generalidades

Un evento de sistema es un evento de entrada externo (generado por un actor externo). Están asociados a las operaciones de sistema.

Un controlador es un objeto de interfase no-usuario responsable de manejar el evento de sistema. Un controlador define el método para la operación del sistema.

La misma clase de controlador debería ser usada para todos los eventos del sistema de un mismo caso de uso, dado que es posible mantener información del estado del caso de uso. Tal información es útil par identificar eventos fuera de secuencia. Diferentes controladores pueden ser usados para diferentes casos de uso.

Un colorario importante del patrón controlador es que los objetos de interfase externa (tales como ventanas, applets) y la capa de presentación no tendrían la responsabilidad de completar los eventos de sistema. Osea, las operaciones de sistema serían manejadas a nivel del dominio del sistema y no a nivel de la capa de aplicación, presentación o interfase.

6- Patrón Polimorfismo.

Problema: *¿Cómo manejar alternativas basadas sobre tipos? ¿Cómo crear componentes de software conectores?*

Alternativas basadas sobre tipos: variaciones condicionales de programas.

Componentes de software conectores: las visiones en relaciones cliente/servidor. ¿Cómo podemos reemplazar un componente de software por otro sin afectar el cliente?

Solución: *Cuando las alternativas o comportamientos relacionados varían por tipo (clase), asignar la responsabilidad para el comportamiento (usando operaciones polimórficas) a los tipos para el cual el comportamiento varía.*

Beneficios:

- Un diseño basado en la asignación de responsabilidades por polimorfismo es fácilmente extensible para manejar nuevas variaciones.

7- Patrón Fabricación Pura.

Problema: *¿Quién tendrá la responsabilidad de realizar una tarea, cuando todo es riesgoso y no queremos violar los patrones de alta cohesión y bajo acoplamiento?*

Solución: *Asignar un conjunto de responsabilidades altamente cohesivos a una clase artificial que no representa nada del dominio del problema, para soportar alta cohesión, bajo acoplamiento y reusabilidad.*

Generalidades:

Una fabricación pura es diseñada con un alto potencial para la reusabilidad y como un medio de garantizar responsabilidades pequeñas y cohesivas.

Una fabricación pura es usualmente particionada teniendo en cuenta las funciones, y de esta manera, es una clase de objetos centrados en funciones.

Beneficios:

- *Alta cohesión es soportada porque las responsabilidades son asignadas poniendo un conjunto muy específico de tareas relacionadas en una clase.*
- *Poseen reusabilidad potencial, dado que las funciones que posee la clase de fabricación pura tienen utilidad en otras aplicaciones.*

Problema:

- El diseño orientado a objetos se centra en objetos más que en funciones. Las fabricaciones puras son particionadas basadas en las funcionalidades, o sea haciendo clases con conjuntos de funciones. Si abusamos, la fabricación pura nos conducirá a un diseño orientado a procesos o funciones más que implementada en un lenguaje orientado a objetos.

8- Indirección.

Problema: *¿Quién tendrá la responsabilidad de realizar una tarea de evitar el acoplamiento directo? ¿Cómo desacoplar objetos de manera que el acoplamiento permanezca bajo y la reusabilidad potencial permanezca alta?*

Solución: *Asignar la responsabilidad a un objeto intermediario que medie entre otros componentes o servicios dado que él no posee acoplamiento directo.*

El intermediario crea una indirección entre los componentes y los servicios.

Beneficios:

- Bajo acoplamiento

9- No hablar con extraños.

Problema: *¿Quién tendrá la responsabilidad de realizar determinada tarea, para evitar conocer la estructura de objetos indirectos?*

Si un objeto tiene conocimiento de las conexiones y estructuras internas de otros objetos, luego él sufre de alto acoplamiento. Si un objeto cliente tiene que usar un servicio o obtener información de un objeto indirecto ¿Cómo lo hace para no acoplarse conociendo la estructura interna del objeto indirecto?

Solución: *Asignar la responsabilidad a un objeto directo del cliente para colaborar con el objeto indirecto y así evitaremos que el cliente tenga conocimiento acerca del objeto indirecto.*

Así los objetos directos son "familiares del cliente" y los indirectos son "extraños" y un cliente solo puede hablar con "familiares" y no con "extraños".

Beneficios:

- Bajo acoplamiento

Anexo "Visibilidad"

Introducción

Para que un objeto envíe un mensaje a otro objeto, tiene que tener cierta visibilidad sobre el mismo. O sea si un objeto A envía un mensaje a un objeto B, luego el objeto a debe tener cierta visibilidad sobre el objeto B (el objeto A tiene que tener una referencia o un puntero al objeto B).

Cuando estamos creando un diseño de objetos interactuantes, es necesario garantizar la visibilidad para soportar la interacción entre mensajes.

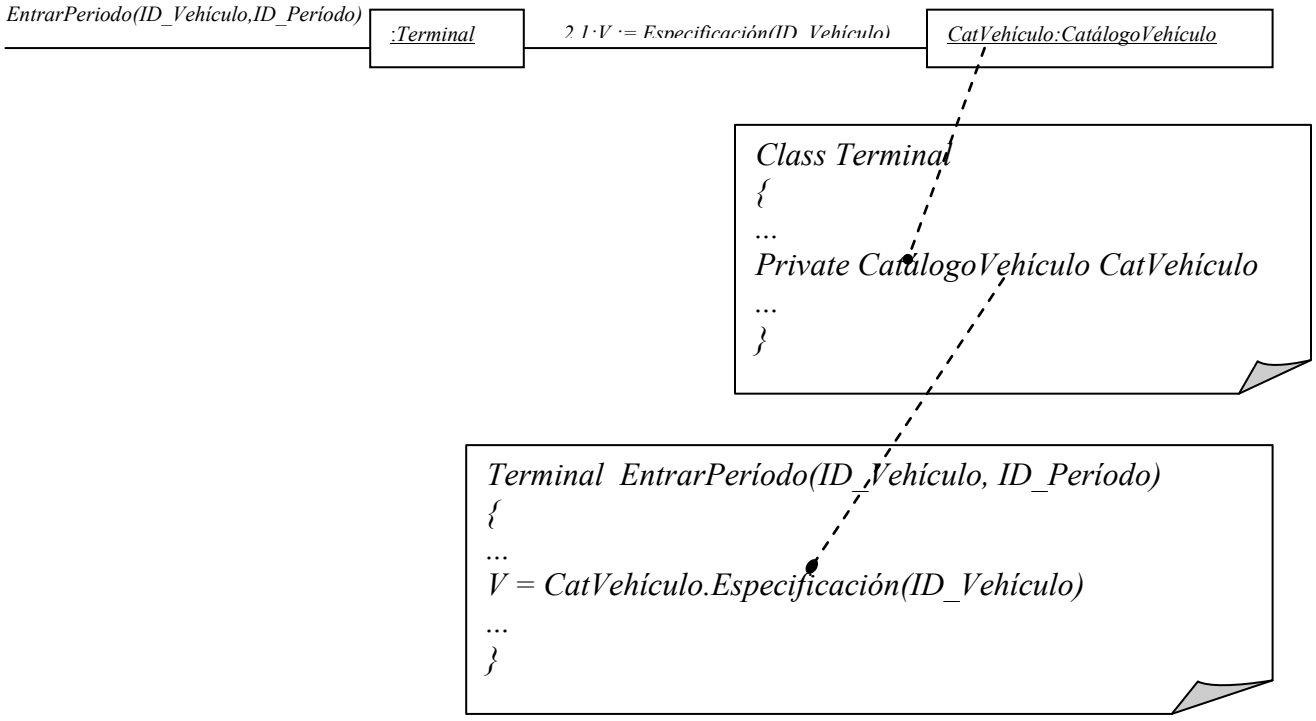
La visibilidad es la habilidad de un objeto de "ver" o tener una referencia a otro objeto. Hay cuatro formas comunes de visibilidad:

- *Visibilidad por atributo.*
- *Visibilidad por parámetro.*
- *Visibilidad por declaración local.*
- *Visibilidad global.*

Visibilidad por Atributo

La visibilidad por atributo de A a B, existe cuando B es un atributo de A.

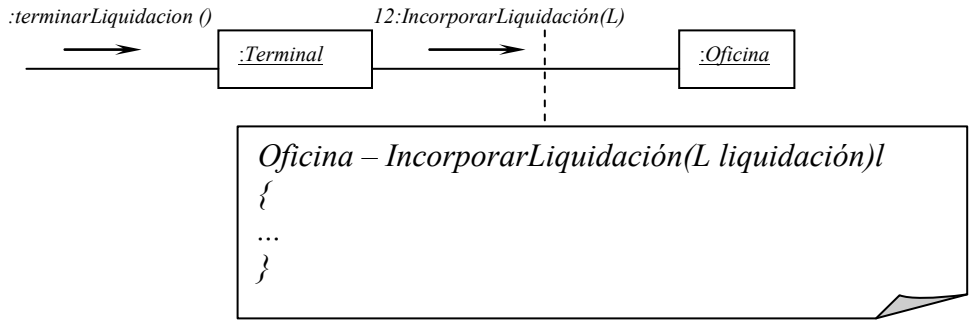
Por ejemplo miremos la siguiente porción de diagrama de colaboración:



Terminal necesita visibilidad por atributo a CatálogoVehículo, porque el diagrama de colaboración "EntrarPeríodo" necesita enviar el mensaje "Especificación".

Visibilidad por Parámetro

La visibilidad por parámetro de A a B, existe cuando B es pasado como un método de A. Es una relación relativamente temporaria, dado que persiste solamente dentro del alcance del método.

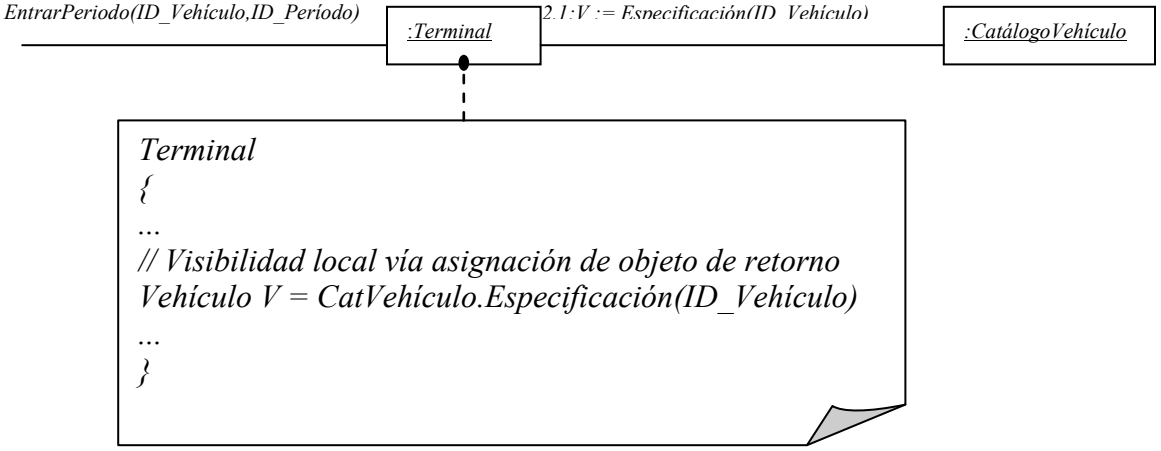


Visibilidad Declarada Localmente

La visibilidad declarada localmente de A a B, existe cuando B es declarada como un objeto local dentro de un método de A. Su persistencia está dentro del alcance del método. Dos casos comunes en los que esta forma de visibilidad es alcanzada son:

- *Crear una nueva instancia local y asignarla a una variable local.*
- *Asignar el objeto de retorno desde una invocación de método a una invocación local.*

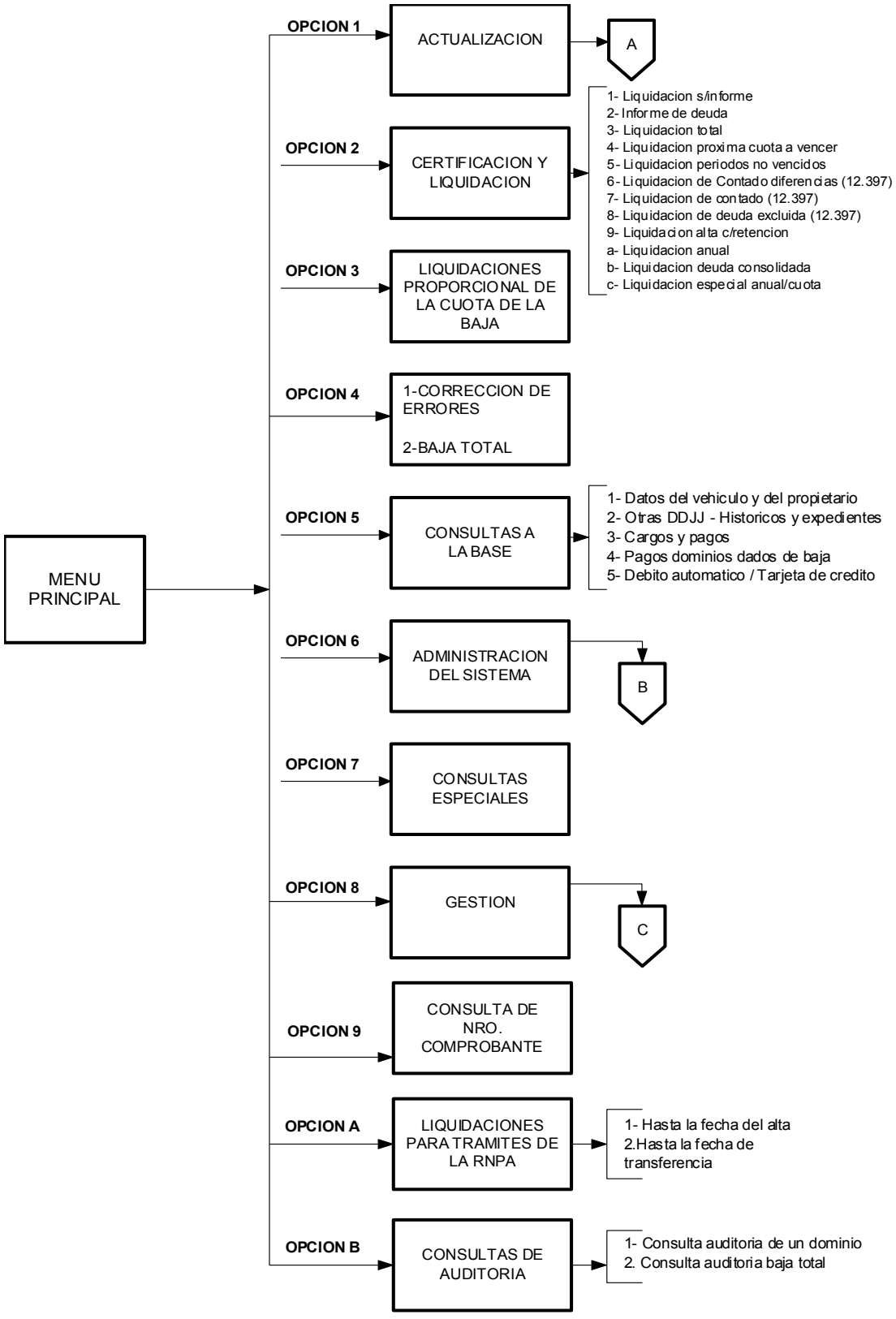
En EntrarPeríodo una variación es:

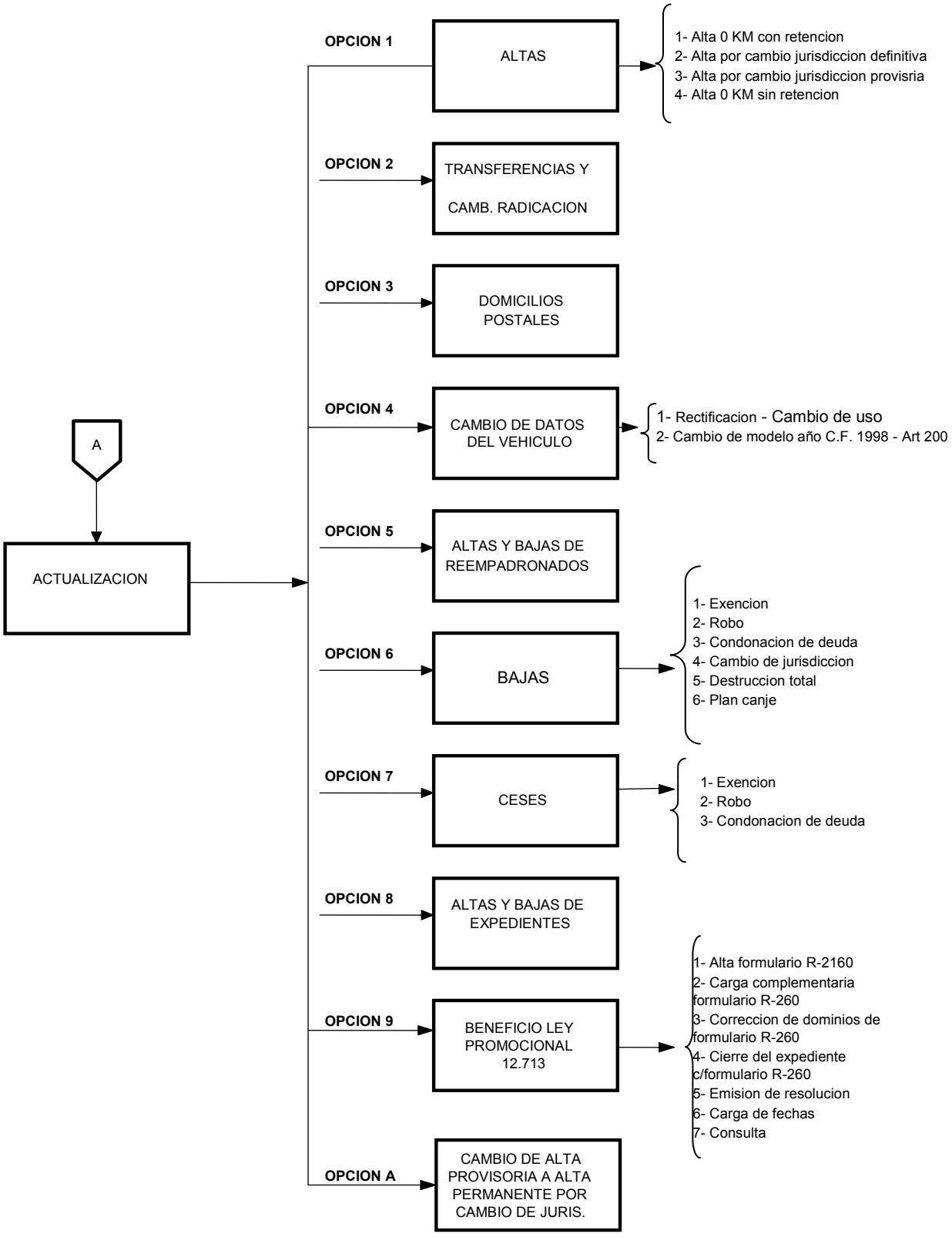


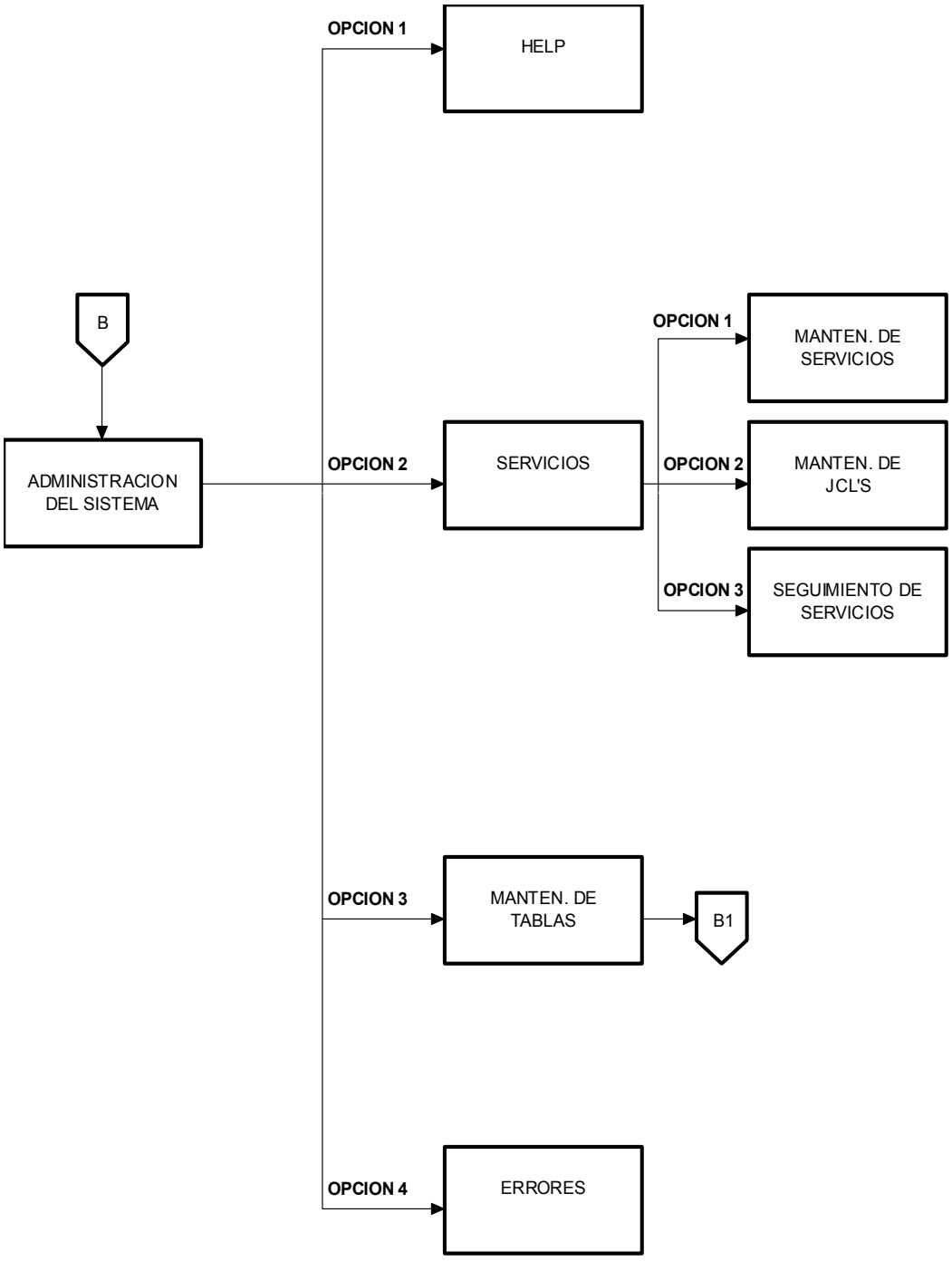
Visibilidad Global.

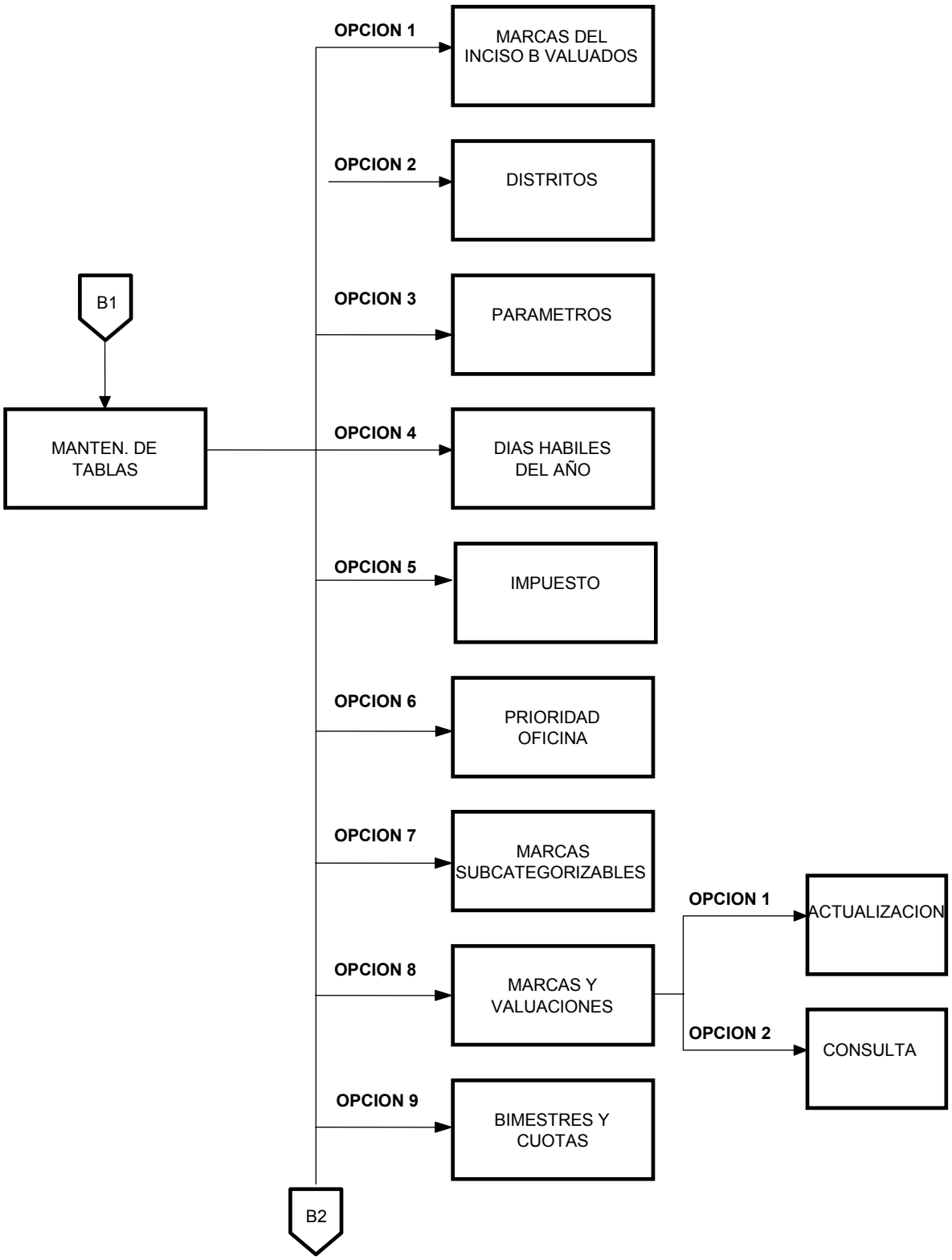
La visibilidad global de A a B, existe cuando B es global a A. Esta visibilidad es relativamente permanente porque su persistencia es a lo largo de que A y B existen.

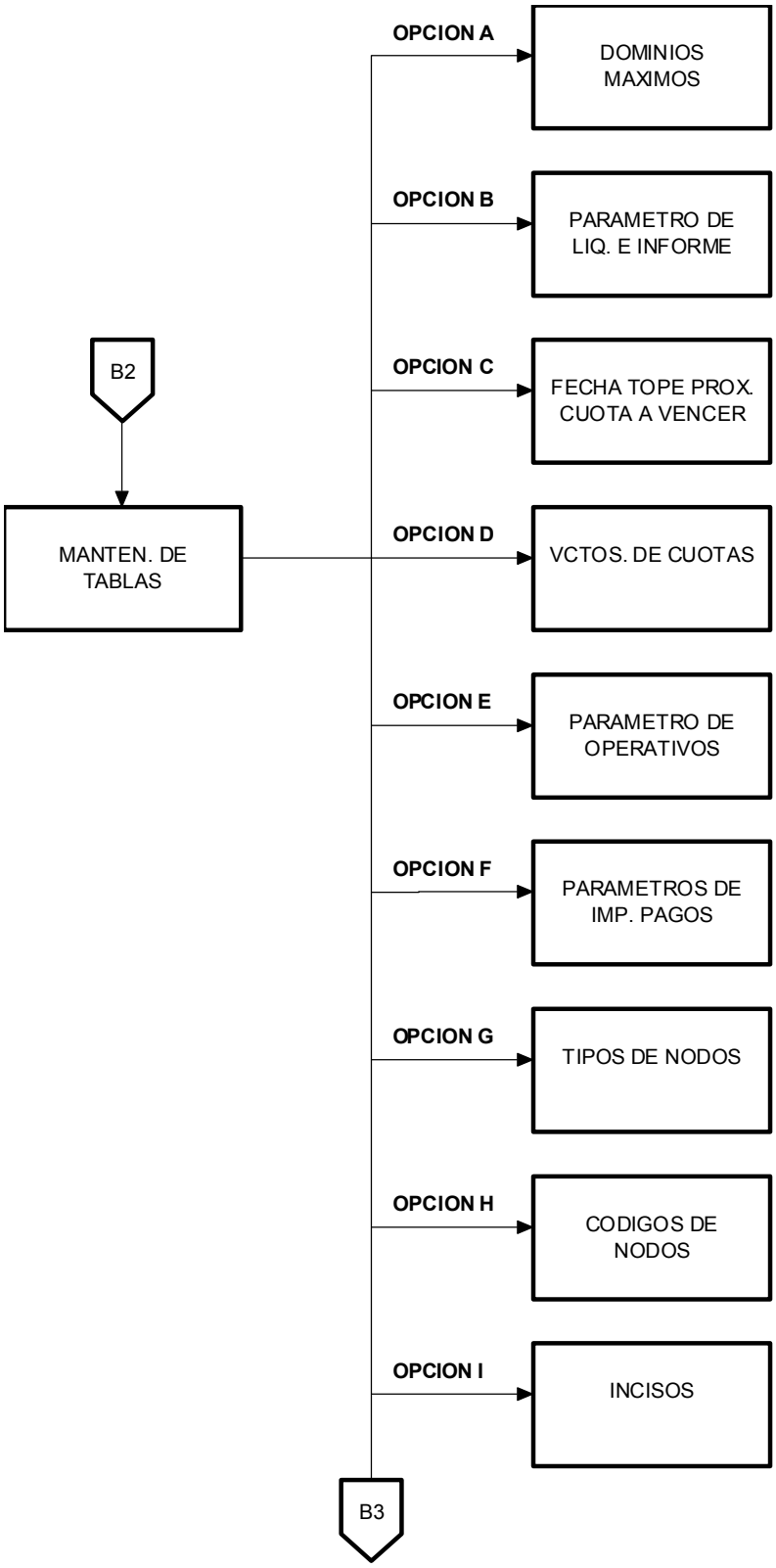
Anexo "Jerarquía del Sistema Online"

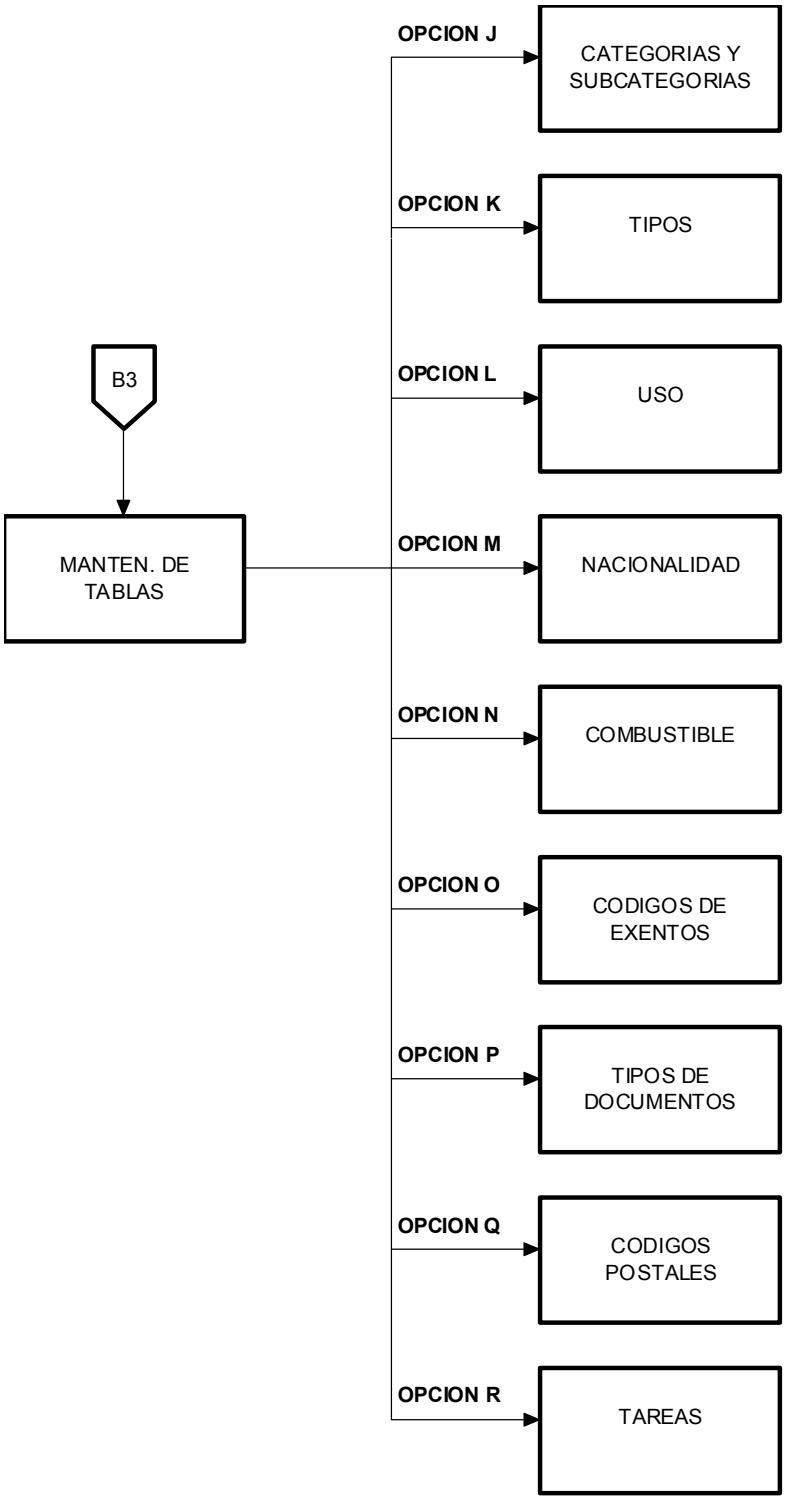


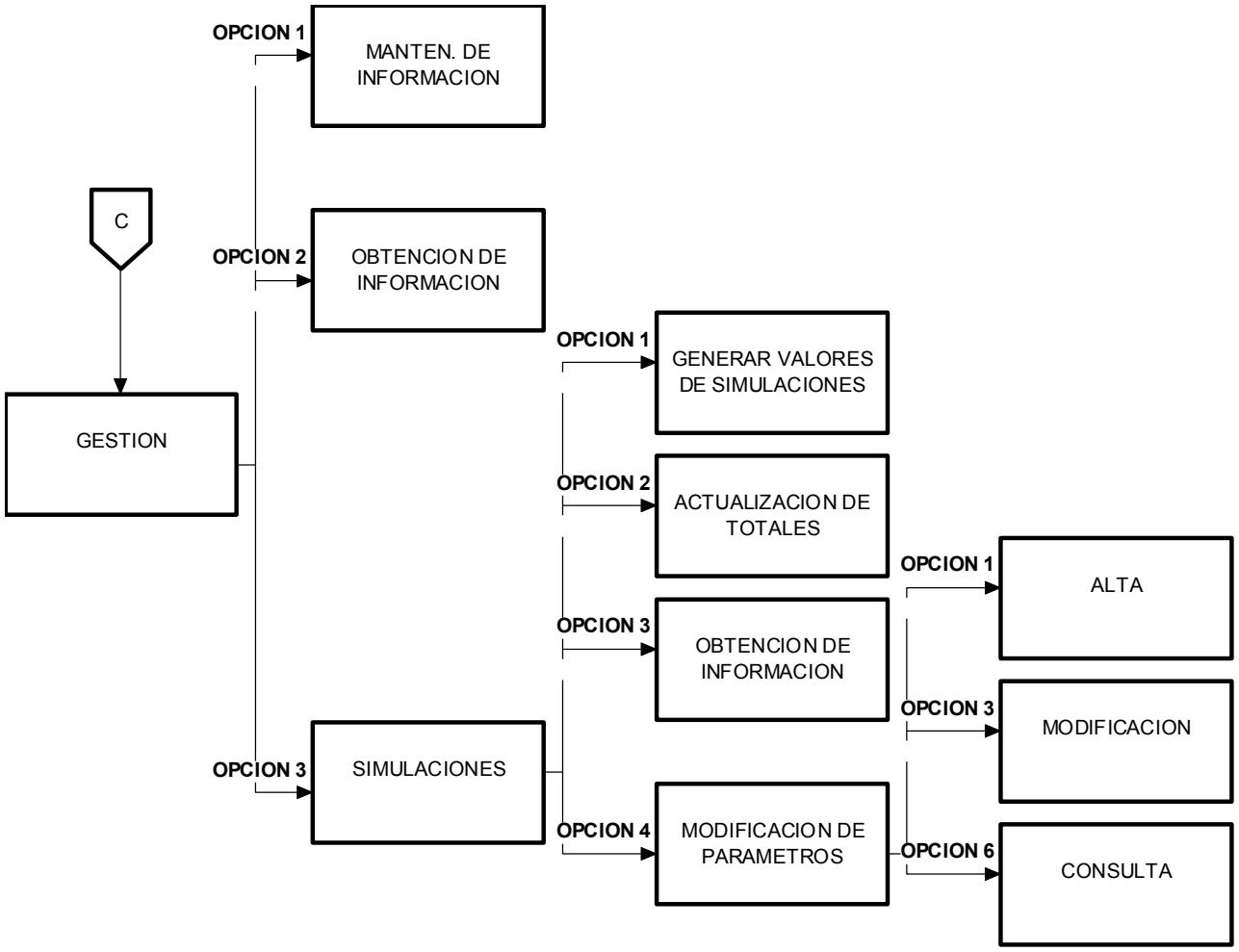

A
ACTUALIZACION
OPCION 1
ALTAS
1- Alta 0 KM con retencion
2- Alta por cambio jurisdiccion definitiva
3- Alta por cambio jurisdiccion provisria
4- Alta 0 KM sin retencion
OPCION 2
TRANSFERENCIAS Y
CAMB. RADICACION
OPCION 3
DOMICILIOS
POSTALES
OPCION 4
CAMBIO DE DATOS
DEL VEHICULO
1- Rectificacion - Cambio de uso
2- Cambio de modelo año C.F. 1998 - Art 200
OPCION 5
ALTAS Y BAJAS DE
REEMPADRONADOS
OPCION 6
BAJAS
1- Exencion
2- Robo
3- Condonacion de deuda
4- Cambio de jurisdiccion
5- Destruccion total
6- Plan canje
OPCION 7
CESES
1- Exencion
2- Robo
3- Condonacion de deuda
OPCION 8
ALTAS Y BAJAS DE
EXPEDIENTES
OPCION 9
BENEFICIO LEY
PROMOCIONAL
12.713
1- Alta formulario R-2160
2- Carga complementaria
formulario R-260
3- Correccion de dominios de
formulario R-260
4- Cierre del expediente
c/formulario R-260
5- Emision de resolucion
6- Carga de fechas
7- Consulta
OPCION A
CAMBIO DE ALTA
PROVISORIA A ALTA
PERMANENTE POR
CAMBIO DE JURIS.


OPCION 1
HELP
B
ADMINISTRACION DEL SISTEMA
OPCION 2
SERVICIOS
OPCION 1
MANTEN. DE SERVICIOS
OPCION 2
MANTEN. DE JCL'S
OPCION 3
SEGUIMIENTO DE SERVICIOS
OPCION 3
MANTEN. DE TABLAS
B1
OPCION 4
ERRORES


B1
MANTEN. DE TABLAS
OPCION 1
MARCAS DEL INCISO B VALUADOS
OPCION 2
DISTRITOS
OPCION 3
PARAMETROS
OPCION 4
DIAS HABILES DEL AÑO
OPCION 5
IMPUESTO
OPCION 6
PRIORIDAD OFICINA
OPCION 7
MARCAS SUBCATEGORIZABLES
OPCION 8
MARCAS Y VALUACIONES
OPCION 1
ACTUALIZACION
OPCION 2
CONSULTA
OPCION 9
BIMESTRES Y CUOTAS
B2


B2
MANTEN. DE TABLAS
OPCION A
DOMINIOS MAXIMOS
OPCION B
PARAMETRO DE LIQ. E INFORME
OPCION C
FECHA TOPE PROX. CUOTA A VENCER
OPCION D
VCTOS. DE CUOTAS
OPCION E
PARAMETRO DE OPERATIVOS
OPCION F
PARAMETROS DE IMP. PAGOS
OPCION G
TIPOS DE NODOS
OPCION H
CODIGOS DE NODOS
OPCION I
INCISOS
B3


B3
MANTEN. DE TABLAS
OPCION J
CATEGORIAS Y SUBCATEGORIAS
OPCION K
TIPOS
OPCION L
USO
OPCION M
NACIONALIDAD
OPCION N
COMBUSTIBLE
OPCION O
CODIGOS DE EXENTOS
OPCION P
TIPOS DE DOCUMENTOS
OPCION Q
CODIGOS POSTALES
OPCION R
TAREAS



Nota:

Esta jerarquía de sistemas del centro de cómputos, suele estar acompañada de los nombres de programas. Aquí se han omitido para no volver más engorrosa la lectura del esquema.

Anexo "Muestra del Análisis de la Liquidación"

Aclaración: *Este análisis no es mostrado en su totalidad ya que lo utilizamos para comparar la metodología de trabajo del centro de cómputos respecto a la de esta tesis. Consideramos que, con esta muestra, el lector podrá tomar conocimiento de la forma en que se realizan los análisis en el centro de cómputos y esto le permitirá comprender mejor las conclusiones del capítulo 17.*

Objeto: *Liquidación e informe de deuda online.*

FILE'S: *Nombres de files asociados a la liquidación*

PANTALLAS: *Nombres de mapas (pantallas) asociadas a la liquidación.*

OPCIONES DE TRABAJO
1- Liquidación sin informe.
2- Informe de deuda (no documento de pago).
3- Liquidación total.
4- Liquidación próxima cuota a vencer.
5- Liquidación deudas año corriente.
6- Liquidación de años prescriptos
7- Liquidación de diferencias
8- Informe de deuda para juzgado (no valido para el pago)
9- Liquidación con retención

PARÁMETROS DEL PROCESO: *Enumeración de los parámetros utilizados en la liquidación.*

VALIDACIÓN DE PANTALLA 1

CAMPO	***VALIDACIÓN***	***LEYENDA POR ERROR***
Opción	*'1' a '8'*	*ERROR EN OPCIÓN*
Dominio	*Letra: alfabética y distinta de ' ', 'I', 'Ñ' y 'O'* *Numero: numérico > 0* *O* *3 letras distintas de 'Ñ'*	*ERROR EN DOMINIO*
Fecha de vcto	*1- Si opción es 4 no se valida.* *2- Puede ser blanco* *3- Si es distinto de blanco:* *01 <= DD <= 31* *01 <= MM <= 12* *AA = año de proc.* *Si MM = 12 puede ser* *AA = año proc. + 1* *DDMMAA >= Fecha de proc.* *Si supera el ultimo día hábil del 3er mes a partir del mes corriente, toma esta fecha como fecha de vcto.*	*FECHA DE VENCIMIENTO ERRÓNEA*

Oficina Solicitante	*Debe existir en AUF-TABLAS con descriptor CLAVE-DISTRI y con MARCA-DISTRI NO = blanco u 'O'*	*OFICINA SOLICITANTE ERRÓNEA*
Liquidador	*Numérico > 0*	*LIQUIDADOR ERRÓNEO*
Casillero	*No se valida*	
Operativo	*1- Si es blanco, forzar 'R00'* *2- Si no debe ser 'R00'*	*OPERATIVO ERRÓNEO*
Solicitud	*No se valida*	
Campos juzg.	*Para Opción 8 debe ser distinto de blanco*	*INGRESE EXPEDIENTE U OFICIO JUDICIAL*

1- Por cualquier error, mandar la leyenda correspondiente y no emitir recibo. Posicionando el cursor en el campo erróneo.
2- Se deberá retener siempre en pantalla el valor de oficina solicitante, operador y letra-chapa.

DOMINIO INEXISTENTE
Si el dominio de pantalla no existe en AUF-MAESTRO, para todas las opciones, se manda a pantalla 1 la leyenda: 'FALTA DECLARACIÓN JURADA - SI DESEA COMPROBANTE PRESIONE PF5', si no se presiona PF5 no se emite formulario, caso contrario se emite el formulario con la leyenda FALTA DECLARACIÓN JURADA y a pantalla AAUM1501 la leyenda CERTIFICADO IMPRESO.

DOMINIO REEMPADRONADO
Si el dominio de pantalla es del tipo 1 letra y 7 números y existe en AUF-MAESTRO con CLAVE-CHAPA-NUEVA distinta de blanco se manda a pantalla AAUM1501 la leyenda: 'DOMINO REEMPADRONADO - SOLICITAR LIQUIDACIÓN CON AAANNN', donde AAANNN es la CLAVE-CHAPA-NUEVA.

DOMINIOS INACTIVOS
1- Si el dominio existe en base con CODIGO-BAJA distinto de blanco continuar con el proceso según opción, pero no se accederá al campo PERIOD-HISTOR, se liquidara la deuda existente, mandando la leyenda a pantalla 1:

'DOMINIO DADO DE BAJA-CANC-DEST-EXENTO-CERTIFICADO IMPRESO'

y a recibo la leyenda (en las mismas líneas en que salen los históricos):

CODIGO-BAJA	***LEYENDA***
'B'	*'** DOMINIO DADO DE BAJA DESDE XX/XXXX. P/REINC. ADJUNTAR DDJJ Y TIT. PROP. **'.*
'C'	*'** DOMINIO CANCELADO POR ROBO DESDE XX/XXXX **'*
'D'	*'** DOMINIO DADO DE BAJA POR DEST. TOTAL O INUTILIZACION. DESDE XX/XXXX **'.*

'0' a '9'	*'** DOMINIO EXENTO DESDE XX/XXXX **'*
'G'	*'** DOMINIO CON PLAN CANJE DESDE XX/XXXX **'*

2- Donde XX/XX es mes-año de FECHA-BAJA.

3- Si el año de FECHA-BAJA es menor al primer año cuota a liquidar de parámetro 1, se completaran los campos según diseño de formulario y en total a abonar llevara todos asteriscos.

CAMBIOS DE JURISDICCIÓN
Solo para opciones 2, 3 y 8:

Si el año de FECHA-ALTA es igual a año de proceso, CODIGO-RECUPE = 'J' o 'P' y no tiene ningún cargo para el periodo año-cuota-desde año-cuota-hasta de parámetro 1.le mandará a pantalla la leyenda 'CERTIFICADO IMPRESO' y a recibo: 'TRIBUTA A PARTIR DEL AÑO SIGUIENTE' donde salen los históricos, mandando todos asteriscos a total a abonar. El
resto de los campos se completan según diseño.

CONSIDERACIÓN PARA CHEQUE RECHAZADO Y TITULO EJECUTIVO
Si para el año-cuota CANTID-EXPEDI = 3, 13, 23, 44, 84, 04, 14, 24, 34, 94 considerar a IMPORT-PAGADO con valor cero.

PRÓXIMA CUOTA A VENCER (PARA LAS OPCIONES 1, 3, 5, 7)
Antes de mostrar y/o liquidar los años-cuotas verificar si el ultimo año-cuota esta en parámetro 2:

1- Si no esta continuar con el proceso.

2- Si esta y la única o la ultima fecha de vcto de esta parámetro es menor a fecha del día continuar con el proceso.

3- Si esta y la única o la ultima fecha de vcto de esta parámetro es mayor o igual a fecha del día: ignorar para el proceso según opción esta cuota ya que solo la podrá liquidar por opción 4 o 6.

DETERMINACIÓN DE LA FECHA DE VENCIMIENTO DE LA LIQUIDACIÓN
(Cuando esta fecha ingreso en blanco en pantalla)

1- Se buscara en el file AUF-TABLAS los días hábiles del año-mes de proceso y del mes siguiente, con el descriptor CLAVE-ANIO-MES.

2- Ubicar el día de proceso en el campo múltiple FECHA-DIAS-HABILE, si no esta hacerlo en la siguiente ocurrencia, si esta ocurrencia es cero o bien no hay mas ocurrencias pasar al siguiente CLAVE-ANIO-MES, con lo que se provoca cambio de mes o inclusive de año.

3- A partir de allí desplazarse 5 ocurrencias, con el día obtenido, con el año-mes de donde se obtuvo el día se forma la fecha de vencimiento de la liquidación.

4- Si por algún motivo no se pudo obtener la fecha de vcto de la liquidación mandar a pantalla el mensaje: 'ERROR EN FECHA VCTO. LIQUIDACIÓN - AVISE A MESA DE AYUDA'.

PROCESO SEGÚN OPCIÓN

1- Liquidación sin informe (opciones 1)

A- Se desplegaran en pantalla 2 todos los años-cuotas que estén comprendidos entre año-cuota-desde y año-cuota-hasta de parámetro 2', y de ellos solamente los que tengan IMPORT-EMITID mayor a IMPORT-PAGADO (en este campo se acumula también los pagos a verificar y las moratorias) (cargos con deuda). Tener en cuenta los años con cuota a transformar, a ignorar según esta parámetro y los años-cuotas especiales de parámetro como los de moratoria, los cuales se informan en la línea 3 (otros conceptos adeudados).

B- Si todos los años-cuotas tienen el IMPORT-EMITID < = a IMPORT-PAGADO o no tiene ningún año-cuota para este periodo, mandar a pantalla 1 la leyenda: 'ESTE DOMINIO NO REGISTRA DEUDA' y no emitir recibo.

D- Si la cantidad de cargos a desplegar superan las 36 ocurrencias, desplegar los primeros 36 cargos en pantalla 2, mandando la leyenda 'HAY MAS CARGOS, SI DESEA VERLOS TIPEE PF8', si el operador tipea PF8 en mostrar la pantalla nuevamente con los restantes cargos, caso contrario se asumirá que no desea verlos.

E- Si la cantidad de cargos a desplegar es < = a 36 ocurrencias, desplegar estos cargos en pantalla 2 sin leyenda.

F- Verificar que en alguna de las dos pantallas (o solo en la primera si no hizo uso de la segunda), haya marcado al menos un cargo con '', si no lo hizo mandar a la pantalla AAUM1501 el mensaje: 'DEBE MARCAR AL MENOS UN AÑO-CUOTA-RETIPEE', caso contrario pasar a PROCESO DE LIQUIDACIÓN.*

G- Se procederá a liquidar solo los años-cuotas marcados con ''.*

H- En pantalla se mostrara FECHA-ANIO, CODIGO-CUOTA, IMPORT-EMITID, IMPORT-PAGADO y deuda original = IMPORT-EMITID - IMPORT-PAGADO.

Aclaración: Aquí continua la descripción de las demás opciones.